FreeFem++ Manual

version 1.44 (Under construction)

http://www.freefem.org
http://www.ann.jussieu.fr/~hecht/freefem++.htm

F. Hecht ¹, O. Pironneau ²,
Université Pierre et Marie Curie,
Laboratoire Jacques-Louis Lions,
175 rue du Chevaleret, PARIS XIII

K. Ohtsuka ³,
Hiroshima Kokusai Gakuin University, Hiroshima, Japan

January 26, 2005

¹mailto:hecht@ann.jussieu.fr
²mailto:pironneau@ann.jussieu.fr
³mailto:ohtsuka@barnard.cs.hkg.ac.jp
## CONTENTS

3.7 splitmesh .................................................. 47
3.8 A Fast Finite Element Interpolator ...................... 48
3.9 Meshing Examples ........................................ 50

### 4 Finite Elements ........................................ 55

4.1 Lagrange finite element .................................. 57
  4.1.1 P0-element ............................................ 57
  4.1.2 P1-element ............................................ 57
  4.1.3 P2-element ............................................ 58
4.2 P1 Nonconforming Element .................................. 58
4.3 Other FE-space ............................................. 60
4.4 Vector valued FE-function ................................ 60
  4.4.1 Raviant-Thomas element ................................. 61
4.5 Problem and solve ......................................... 63
4.6 Parameter Description for solve and problem .......... 63
4.7 Problem definition ......................................... 64
4.8 Numerical Integration ..................................... 65
4.9 Variational Form, Sparse Matrix, Right Hand Side Vector .......................... 66
4.10 Interpolation matrix .................................... 69
4.11 Finite elements connectivity ............................ 70

### 5 Visualization ........................................... 71

5.1 Plot ....................................................... 71
5.2 link with gnuplot ......................................... 73
5.3 link with medit .......................................... 73

### 6 Algorithms ............................................... 77

6.1 conjugate Gradient/GMRES .............................. 77
6.2 Optimization ............................................. 79

### 7 Mathematical Models .................................... 81

7.1 Static Problems .......................................... 81
  7.1.1 Soap Film ............................................. 81
  7.1.2 Electrostatics .......................................... 83
  7.1.3 Aerodynamics ......................................... 85
  7.1.4 Error estimation .................................... 86
  7.1.5 Periodic ................................................ 88
  7.1.6 Poisson with mixed boundary condition ............ 90
  7.1.7 Adaptation with residual error indicator .......... 92
7.2 Elasticity ................................................ 93
  7.2.1 Fracture Mechanics ................................... 96
7.3 Nonlinear Static Problems ................................ 99
  7.3.1 Non linear conjugate gradient algorithm .......... 100
  7.3.2 Newton Raphson algorithm ............................ 102
7.4 Eigenvalue Problems .................................... 103
7.5 Evolution Problems ..................................... 107
  7.5.1 Mathematical Theory on time difference .......... 108
CONTENTS

7.5.2 Convection .................................................. 110
7.5.3 Two-dimensional Black-Scholes equation ................. 112

7.6 Navier-Stokes Equation ........................................ 115
7.6.1 Stokes and Navier-Stokes .................................. 115
7.6.2 Uzawa Conjugate Gradient .................................. 120
7.6.3 NSUzawaCahouetChabart.edp .............................. 121

7.7 Domain decomposition .......................................... 122
7.7.1 Schwarz Overlap Scheme .................................... 122
7.7.2 Schwarz non Overlap Scheme ............................... 124
7.7.3 Schwarz-gc.edp ............................................. 126

7.8 Fluid/Structures Coupled Problem ............................. 128
7.9 Transmission Problem ......................................... 131
7.10 Free Boundary Problem ....................................... 133
7.11 nonlinear-elas.edp ............................................. 136

8 Parallel version experimental ................................. 143
8.1 Schwarz in parallel .............................................. 143

9 Graphical User Interface ....................................... 145
9.1 FreeFem++-cs .................................................. 145
9.2 FFedit .......................................................... 146
9.2.1 Installation of Tcl and Tk .................................. 146
9.2.2 Description .................................................. 147
9.2.3 Cygwin version .............................................. 150
9.2.4 Installation of Cygwin ...................................... 150
9.2.5 Compilation and Installation of FreeFem++ under Cygwin .................................................. 151
9.2.6 Compilation and Installation of tcl8.4.0 and tk8.4.0 under Cygwin .............................................. 152
9.2.7 Use ............................................................ 154

10 Mesh Files ........................................................ 155
10.1 File mesh data structure ...................................... 155
10.2 bb File type for Store Solutions ............................. 156
10.3 BB File Type for Store Solutions ............................ 156
10.4 Metric File ...................................................... 157
10.5 List of AMFMT, AMDBA Meshes ............................. 157

11 Add new finite element ......................................... 161
11.1 Some notation .................................................. 161
11.2 Which class of add ............................................. 162
11.3 How to add ...................................................... 165

A Table of Notations ............................................... 167
A.1 Generalities ..................................................... 167
A.2 Sets, Mappings, Matrices, Vectors .......................... 167
A.3 Numbers ........................................................ 168
A.4 Differential Calculus .......................................... 168
A.5 Meshes ........................................................... 169
A.6 Finite Element Spaces ......................................... 169
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1</td>
<td>Keywords</td>
<td>171</td>
</tr>
<tr>
<td>B.2</td>
<td>The bison grammar</td>
<td>171</td>
</tr>
<tr>
<td>B.3</td>
<td>The Types of the languages, and cast</td>
<td>175</td>
</tr>
<tr>
<td>B.4</td>
<td>All the operators</td>
<td>180</td>
</tr>
<tr>
<td>B.5</td>
<td>History of the software</td>
<td>185</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

A partial differential equation is a relation between a function of several variables and its (partial) derivatives. Many problems in physics, engineering, mathematics and even banking are modeled by one or several partial differential equations.

**Freefem** is a software to solve these equations numerically. As its name says, it is a free software (see copyright for full detail) based on the Finite Element Method. This software runs on all unix OS (with g++ 2.95.2 or better and X11R6), on Window95, 98, 2000, NT, XP, on MacOS 9 and X.

Many phenomena involve several different fields. Fluid-structure interactions, Lorenz forces in liquid aluminium and ocean-atmosphere problems are three such systems. These require approximations of different degrees, possibly on different meshes. Some algorithms such as Schwarz’ domain decomposition method also require data interpolation on different meshes within one program. **freefem++** can handle these difficulties, i.e. *arbitrary finite element spaces on arbitrary unstructured and adapted meshes*.

The characteristics of **freefem++** are:

- A large variety of finites elements: linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements, elements of a non-scalar type, mini-element, ...

- Automatic interpolation of data on different meshes to an over mesh, store the interpolation matrix.

- Linear problems description (real or complex) thanks to a formal variational form, with access to the vectors and the matrix if needed.

- Includes tools to define discontinuous Galerkin formulations (please refer to the following keywords: “jump”, “average”, “intalledges”).

- Analytic description of boundaries. When two boundaries intersect, the user must specify the intersection points.
• Automatic mesh generator, based on the Delaunay-Voronoï algorithm. Inner points density is proportional to the density of points on the boundary [6].

• Metric-based anisotropic mesh adaptation. The metric can be computed automatically from the Hessien of a solution [7].

• Solvers: LU, Cholesky, Crout, CG, GMRES, UMFPACK linear solver, eigenvalue and eigenvector computation.

• Online graphics, C++-like syntax.

• Many examples: Navier-Stokes, elasticity, Fluid structure, Schwarz’s domain decomposition method, Eigen value problem, residual error indicator, ...

• Parallel version using mpi

1.1 History

The project has evolved from MacFem, PCfem, written in Pascal. The first C version was freefem 3.4; it offered mesh adaptation on a single mesh. A thorough rewriting in C++ led to freefem+ 1.2.10, which also included interpolation over multiple meshes (functions defined on one mesh can be used on any other mesh). Implementing the interpolation from one unstructured mesh to another was not easy because it had to be fast and non-diffusive; for each point, one had to find the containing triangle. This is one of the basic problems of computational geometry (see Preparata & Shamos [13] for example). Doing it in a minimum number of operations was a challenge. Our implementation was $O(n \log n)$ and based on a quadtree.

We are now introducing freefem++, an entirely new program written in C++ and based on bison for a more adaptable freefem language.

The freefem language allows for a quick specification of any partial differential system of equations. The language syntax of freefem++ is the result of a new design which makes use of the STL [21], templates and bison for its implementation. The outcome is a versatile software in which any new finite element can be included in a few hours; but a recompilation is then necessary. The library of finite elements available in freefem++ will therefore grow with the version number. So far we have linear and quadratic Lagrangian elements, discontinuous P1 and Raviart-Thomas elements.

1.2 Getting Started

We explain how freefem++ solve the problem Poisson: For a given function $f(x, y)$, find a function $u(x, y)$ satisfying

$$ - \Delta u(x, y) = f(x, y) \quad \text{if } (x, y) \text{ is in } \Omega, \quad \Delta u = \partial^2 u / \partial x^2 + \partial^2 u / \partial y^2, \quad (1.1) $$

$$ u(x, y) = 0 \quad \text{if } (x, y) \text{ is on } \partial \Omega \quad (1.2) $$
The following example shows \texttt{freefem++} program solving $u$ when $f(x,y) = xy$ (see 5th line) and $\Omega$ is the unit disk. The boundary $C = \partial \Omega$ is

$$C = \{(x,y) \mid x = \cos(t), y = \sin(t), 0 \leq t \leq 2\pi\} \quad \text{(see 1st line)}$$

The domain will be on the left side of the oriented boundary by the parameter $t$. As illustrated in Fig. 1.2, we can see the isovalue of $u$ by using \texttt{plot} (see 13th line).

\begin{figure}[h]
\centering
\includegraphics[width=0.45\textwidth]{mesh.png} \quad \includegraphics[width=0.45\textwidth]{iso.png}
\caption{mesh $\mathbf{Th}$ by \texttt{build($\mathbf{C}(50)$)} \quad isovalue by \texttt{plot($u$)}}
\end{figure}

\textbf{Example 1}

\begin{verbatim}
1: border C(t=0,2*pi){x=cos(t); y=sin(t); label=1;}
2: mesh Th = buildmesh (C(50));
3: fespace Vh(Th,P2);
4: Vh u,v;
5: func f= x*y;
6: problem Poisson(u,v,solver=LU) =
7:  int2d(Th)(dx(u)*dx(v) + dy(u)*dy(v)) // bilinear part
8:  - int2d(Th)( f*v) // right hand side
9:  + on(1,u=0) ; // Dirichlet boundary condition
10: 
11: real cpu=clock();
12: Poisson; // SOLVE THE PDE
13: plot(u);
14: cout << " CPU time = " << clock()-cpu << endl;
\end{verbatim}

\subsection{1.2.1 FEM by \texttt{freefem++}}

The example shows \texttt{freefem++} covers easily all standard step in FEM (finite element method). We explain how they are done by \texttt{freefem++} in a step-by-step manner.

\textbf{Step 1: Mesh Generation}

\textbf{1st line}: the boundary $\Gamma$ are described analytically (by opposition to CSG) as stated before. In the case $\Gamma = \sum_{j=0}^J \Gamma_j$ with curves $\Gamma_j$, then the user must specify the intersection points
in case two boundaries intersect. By the use of the keyword “label” such as
\begin{verbatim}
  border Gamma1(t=a1,b1) { x=...; y=...; label=1; }
  ...
  border GammaJ(t=aJ,bJ) { x=...; y=...; label=1; }
\end{verbatim}
the user can refer to \( \Gamma \) by the number “1”. (examples are in Section [3.9]).

2nd line: the triangulation \( T_h \) of \( \Omega \) is automatically generated by “buildmesh(C(50))” giving 50 points on \( C \) as in Fig. 1.1. Automatic mesh generation is based on the Delaunay-Voronoi algorithm. Refinement of the mesh are done by increasing the points on \( \Gamma \), for example, “buildmesh(C(100))”, because inner vertices are determined by the density of points on the boundary.

The symbol \( T_h \) (Th in freefem++) shows a family \( \{ T_k \}_{k=1,...,n_t} \) of triangles in Fig. 1.1 with the size \( h \) of the mesh. Here \( n_t \) stands for the number of triangles in \( T_h \). If \( \Omega \) is not polygonal domain, a “skin” remains between the exact domain \( \Omega \) and its approximation \( \Omega_h = \bigcup_{k=1}^{n_t} T_k \). However, we notice that all corners of \( \Gamma_h = \partial \Omega_h \) are on \( \Gamma \).

\textbf{Step2: Making finite element space}

3rd line: “fespace Vh(Th,P2)” makes the continuous Finite Element SPACE
\begin{equation}
V_h(Th,P_2) = \left\{ w(x,y) \mid w(x,y) = \sum_{k=0}^{M-1} w_k \phi_k(x,y), w_k \text{ are real numbers} \right\}
\end{equation}
where \( P_2 \) indicate \( \phi_k \) is a polynomial of degree \( \leq 2 \), that is, in each \( T_k \),
\[ \phi_k(x,y) = \alpha_1 + \alpha_2 x + \alpha_3 y + \alpha_4 x^2 + \alpha_5 x y + \alpha_6 y^2 \]
and the constants \( \alpha_1, \cdots, \alpha_6 \) are defined by its values at the vertices of \( T_k \) and their middle points that continuous in \( \Omega \). Here \( w_k \) are called the degree of freedom of \( w \) and \( M \) the number of the degree of freedom. Already freefem++ implemented \( P_0, P_1, P_2, RT_0, P_{1nc}, P_{1dc}, P_{2dc}, P_{1b} \)-elements. The user can easily add a part of arbitrary degree elements to freefem++ , so the available finite elements will differ with the version.

\textbf{Step3: Setting the problem}

4th line: “Vh u” declare that \( u \) is approximated through the use of the basis functions \( \phi_k \) in \( V_h \), that is,
\[ u(x,y) \simeq u_k(x,y) = \sum_{k=0}^{M-1} u_k \phi_k(x,y) \]

5th line: the given function \( f \) is defined analytically using the keyword \texttt{func}.

6th-9th lines: the formulation of (1.1) and (1.2) are done. Multiplying (1.1) by \( v(x,y) \) and integrating the result over \( \Omega \), we have
\[ -\int_\Omega v \Delta u \, dx\,dy = \int_\Omega v f \, dx\,dy \]
Then, by Green’s formula, the problem Poisson is translated into finding \( u \) such that
\begin{align}
  a(u,v) - \ell(f,v) &= 0 \quad \text{for all } v \quad \text{(1.4)} \\
  a(u,v) &= \int_\Omega \nabla u \cdot \nabla v \, dx\,dy \quad \ell(f,v) = \int_\Omega f v \, dx\,dy \quad \text{(1.5)}
\end{align}
1.2. GETTING STARTED

satisfying \( v = 0 \) on \( \partial \Omega \). In \texttt{freefem++} the problem \texttt{Poisson} is declared by

\[
\texttt{Vh u,v; problem Poisson(u,v) =}
\]

and (1.4) is expressed by symbols \( \text{dx}(u) = \partial u / \partial x \), \( \text{dy}(u) = \partial u / \partial y \) and

\[
\int_{\Omega} \nabla u \cdot \nabla v \, dx \, dy \rightarrow \text{int2d}(\text{Th}) \left( \text{dx}(u) \cdot \text{dx}(v) + \text{dy}(u) \cdot \text{dy}(v) \right)
\]

\[
\int_{\Omega} f v \, dx \, dy \rightarrow \text{int2d}(\text{Th}) \left( f * v \right) \quad \text{(notice here, } u \text{ is unused)}
\]

In \texttt{freefem++} the first and second formulas just above must be distinguished each other. Because, the linear system to be solved are created from substituting \( u_h \) for \( u \) and \( \phi_i \) for \( v \) in (1.4).

\[
A_{ij} u_j - F_i = 0 \quad i,j = 0, \ldots, M - 1; \quad F_i = \int_{\Omega} f \phi_i \, dx \, dy \quad (1.6)
\]

and the solution \( u_h = \sum_{j=0}^{M-1} u_j \phi_j \) must satisfy “\( u_h = 0 \) on \( \Gamma_h \approx C \)”\). The matrix \( A = (A_{ij}) \) is called \textit{stiffness matrix} and is modified from

\[
A^0 = \{A^0_{ij}\}, \quad A^0_{ij} = \int_{\Omega} \nabla \phi_j \cdot \nabla \phi_i \, dx \, dy \quad i,j = 0, \ldots, M - 1 \quad (1.7)
\]

to ensure \( u = 0 \) on \( C \) by “\(+on\{1,u=0\}\) ” in 9th line. If you want use the symbol “\( C \)”. Such as “\(+on\{C,u=0\}\)” (9th line), then the user do not use the keyword “label”.

we can create directly the stiffness matrix \( A \) in (1.6) by

\[
\texttt{varf a(u,v) = int2d(Th) ( dx(u) * dx(v) + dy(u) * dy(v) ) + on(C,u=0) ;}
\]

matrix \( A=a(\text{Vh},\text{Vh}) ; \) \quad // \quad \textit{stiffness matrix, see (1.7)}

(see \texttt{Example 2}). The vector \( B \) in (1.6) is called \textit{load matrix} , and also we get it:

\[
\texttt{varf b([v],[f]) = int2d(Th) (v*f) ;}
\]

\[
\texttt{matrix B=b(\text{Vh},\text{Vh}) ;}
\]

The linear system (1.6) is solved by a Gauss LU factorisation. You can declare the solver of (1.6), for example,

\[
\texttt{Vh u,v; problem Poisson(u,v,solver=CG) =}
\]

means that (1.6) will be solved by Conjugate Gradient method.

Step4: Solving and visualization

11th line: the current time is stored into the real-valued variable \( \texttt{cpu} \).

12th line: the problem is solved by calling its name.

13th line: the visualization is done as illustrated in Fig. 1.2 (see Section 5.1 for zoom, postscript and other commands).

14th line: the time in calculation is outputed into your console (= default of standard output) using C++-like syntax. The user need not study C++ for using \texttt{freefem++} , because C++-like syntax is used for input/output, loops, flow-controls etc.
1.2.2 Features of freefem++

The language it defines is typed, polymorphic and reentrant with macro generation (see §7.11). Every variable must be typed and declared in a statement; each statement separated from the next by a semicolon ‘;’.

For purposes of explanation, we used \( \mathcal{T}_h \) \( (\text{Th}) \), \( V_h \) \( (\text{Vh}) \), unknown function \( u \) \( (\text{u}) \), test functions \( v \) \( (\text{v}) \) and the problem Poisson, etc. (the term inside the parentheses are symbols in freefem++ programming), but you can use any name except reserved words and names already used. Reserved words are shown in blue. \( \pi \), \( x \), \( y \), label, solver are reserved variables. It is allowed (although not advisable) to redefine these variables, so they will not be highlighted again in the following example programs.

In each step, the independence in freefem++ programming is very high as stated below.

- For example, by changing 1st and 2nd lines as following, we can solve (1.1) and (1.2) in L-shape domain with \( \Gamma = \sum_{j=1}^{6} \Gamma_j \).
  
  ```cpp
  border G1(t=0,1) {x=t;y=0; label=1;} // \Gamma_1
  border G2(t=0,0.5) {x=1;y=t; label=1;} // \Gamma_2
  border G3(t=0,0.5) {x=1-t;y=0.5; label=1;} // \Gamma_3
  border G4(t=0.5,1) {x=0.5;y=t; label=1;} // \Gamma_4
  border G5(t=0.5,1) {x=1-t;y=1; label=1;} // \Gamma_5
  border G6(t=0,1) {x=0;y=1-t; label=1;} // \Gamma_6
  mesh Th = buildmesh ( G1(6)+G2(4)+G3(4)+G4(4)+G5(4)+G6(6));
  ```

- In Step 3, you can control where the solution will be approximated. If you write “\( \text{Vh(Th, P1)} \),” in 3rd line, you can get \( P_1 \)-approximation. The machine time by \( P_1 \)-element will be faster than \( P_2 \)-element and the storage is less.

- In Step 4, you can change the equation and boundary conditions easily. For example, if you want solve
  
  \[
  -\text{div}(k(x,y)\nabla u(x,y)) = f(x,y) \quad \text{in } \Omega \\
  u(x,y) = 0 \quad \text{if } (x,y) \text{ on } \Gamma
  \]
  
  you only write as follows

  ```cpp
  func f= ...;
  func k= 1+sin(2*pi*x)*cos(2*pi*y);
  problem Poisson(u,v) =
    int2d(Th) ( k* dx(u)*dx(v) + dy(u)*dy(v) )
    - int2d(Th) ( f*v )
    + on(1,u=0);  
  ```

  The user can use FE-function as the given function \( f \) (see Section 2.6.2), for example, obtained function \( u \) in Example [1]. In freefem++ programming, the easy reuse of the obtained results is important feature.

- The user can easily compare between mathematical modelling and freefem++ program.
1.3 Projectoin or Interpolation

For a finite element space \( V_h \), \( P_1 \)-projection \( \Pi_h \) is defined by

\[
\Pi_h f = f(q^1)\phi_1 + f(q^2)\phi_2 + \cdots + f(q^{n_v})\phi_{n_v}
\]

for all continuous functions \( f \). In \texttt{freefem++} we can easily create the projection \( f_h(=f_h) \) by

\[
V_h f_h = f(x,y); \\
\Pi_h \text{ is also called } P_1 \text{-interpolate.}
\]

1.4 Matrix and Vector

Here, we show how to get the stiffness matrix using \texttt{freefem++} The first command \texttt{varf} is to define the variational formula.

**Example 2** Here we solve the same problem (1.1) and (1.2) using matrix. For purposes of explanation, we change mesh size and use \( P_1 \)-element:

1: \texttt{border} C(t=0,2*pi) { x = cos(t); y = sin(t); }
2: \texttt{mesh} Th = buildmesh(C(7)); \hfill // changed from Example 1
3: \texttt{fespace} Vh(Th,P1);
4: \texttt{Vh u,v,f,F;}
5: \texttt{varf a(u,v) = int2d(Th) (dx(u)*dx(v) + dy(u)*dy(v))}
6: \quad + on(C,u=0); \hfill // see (1.7)
7: \texttt{varf b([v],[f]) = int2d(Th)(v*f)};
8: 9: \texttt{f = x*y; \hfill // interpolate (x,y) \leftarrow x*y function}
10: \texttt{matrix A=a(Vh,Vh);} \hfill // stiffness matrix, see (1.6)
11: \texttt{matrix B=b(Vh,Vh);} \hfill // load vector, see (1.6)
12: \texttt{F[]=B*f[];}
13: \texttt{cout "F=
}
14: \texttt{cout "A=
}
15: \texttt{u[]=A^-1*F[];} \hfill // solve \( A U_h = F \), see (1.6)
16: \texttt{plot(u);} \\

We get the mesh \( T_h = \{ T_1, \ldots, T_7 \} \) (see Fig. 1.3).

**Note 1** In what follows, we denote the vertices by \( q^i \), \( i = 1, \ldots, 8 \), the number of vertices by \( n_v \), the number of triangles by \( n_t \). For each triangle \( T_k \in T_h \), we index the vertices by \( q^{k_1}, q^{k_2}, q^{k_3} \) and denote the edges by \( [q^{k_1}, q^{k_2}], [q^{k_2}, q^{k_3}], [q^{k_3}, q^{k_1}] \), that is, \( [q^i, q^j] \) is the segment connecting \( q^i \) and \( q^j \). We denote the number of edges \( [q^i, q^j] \) by \( n_e \) for all \( q^i, q^j \partial \Omega_h \), \( \Omega_h = \sum_{k=1}^7 T_k \). Here \( n_v = 8, n_t = 7, n_e = 7 \).

The function \( v \) in “\( V_h \)” is expressed

\[
v(x) = v_1\varphi_1(x) + \cdots + v_{n_v}\varphi_{n_v}(x)
\]

using the hat functions \( \varphi_j, j = 1, \ldots, n_v \) (see Fig. 1.4). Here the \( j \)-th hat function \( \varphi_j \) associated with \( j \)-th vertex \( q^j \) is defined in the following way:
1. \( \varphi_j \) is continuous function on \( \Omega_h \).

2. \( \varphi_j \) is linear on each triangle \( T_k, k = 1, \ldots, n_t \) of “\( Th \)”.

3. \( \varphi_j(q^i) = \delta_{ji} \) where \( q^i \) denotes the \( i \)-th vertex, for all \( i = 1, \ldots, n_v \).

Here \( \delta_{ij} \) is the Kronecker symbol.

**Note 2** Other finite element spaces in freefem++ are explained in Section 4.

**Note 3** For an element \( v = v_1 \phi_1 + \cdots + v_M \phi_M \) in a finite element space \( V_h \), we get the column vector \( \{v\} \)

\[
\{v\} = \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix} \text{ in freefem++}
\]

Theoretically, it is natural to use the finite element space

\[ H^1_{0h} = \{ v \in V_h(T_h, P_1) \left| \phi_i(x) = 0 \text{ if } q^i \in \partial \Omega_h \} \]

Let \( I_\Omega \) be the set of indices \( i \) of all internal vertices of the mesh \( \forall h \). In this example, \( I_\Omega = \{6\}. 

The stiffness matrix \( A \) in 10th line is:

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
10^{30} & -0.31 & 0 & 0 & -0.46 & -0.51 & 0 & 0 \\
-0.31 & 10^{30} & -0.23 & 0 & 0 & -0.71 & 0 & 0 \\
0 & -0.23 & 10^{30} & -0.31 & 0 & -0.71 & 0 & 0 \\
0 & 0 & -0.31 & 10^{30} & 0 & -0.51 & -0.46 & 0 \\
-0.46 & 0 & 0 & 0 & 10^{30} & -0.35 & 0 & -0.54 \\
-0.51 & -0.71 & -0.71 & -0.51 & -0.35 & 3.47 & -0.35 & -0.30 \\
0 & 0 & 0 & -0.46 & -0.35 & 10^{30} & -0.54 \\
0 & 0 & 0 & 0 & -0.54 & -0.30 & -0.54 & 10^{20}
\end{bmatrix}
\] (1.8)
that is
\[
A_{ij} = \int_{\Omega_h} \nabla u_j \cdot \nabla u_i \quad \text{if} \ i \neq j, \ i = j \in I_\Omega \tag{1.9}
\]
\[
A_{ij} = E \quad (E = 10^{30}) \quad \text{if} \ j \in I_\Omega. \tag{1.10}
\]

The load matrix $F_T$ is:
\[
\begin{pmatrix}
-0.020 & -0.037 & 0.037 & 0.020 & 0.064 & 0 & -0.064 & 1. \times 10^{-17}
\end{pmatrix}
\]

For $i \notin I_\Omega$,
\[
Eu_i + \sum_{i \neq j} A_{ij} u_j = b_i
\]

which means that
\[
u_i = (b_i - \sum_{i \neq j} A_{ij} u_j) \times E^{-1} \approx 10^{-30} \approx 0
\]

Mathematical results indicate that the Poisson equation with Neumann boundary condition has not unique solution, whose weak form is same to (1.1) except the boundary condition:
\[
\int_{\Omega} \nabla u \cdot \nabla v = \int_{\Omega} f v dx
\]

without pernarization $E$. Then the stiffness matrix is created by
\[
\text{varf } a(u, v) = \text{int2d}(Th)(dx(u) * dx(v) + dy(u) * dy(v))
\]
\[
\text{matrix } A=a(Vh, Vh); \quad \text{stiffness matrix}
\]

and the obtained stiffness matrix is the following
\[
\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1.29 & -0.31 & 0 & 0 & -0.46 & -0.51 & 0 & 0 \\
-0.31 & 1.26 & -0.23 & 0 & 0 & -0.71 & 0 & 0 \\
0 & -0.23 & 1.26 & -0.31 & 0 & -0.71 & 0 & 0 \\
0 & 0 & -0.31 & 1.29 & 0 & -0.51 & -0.46 & 0 \\
-0.46 & 0 & 0 & 0 & 1.35 & -0.35 & 0 & -0.54 \\
-0.51 & -0.71 & -0.71 & -0.51 & -0.35 & 3.47 & -0.35 & -0.30 \\
0 & 0 & 0 & -0.46 & 0 & -0.35 & 1.35 & -0.54 \\
0 & 0 & 0 & 0 & -0.54 & -0.30 & -0.54 & 1.38
\end{pmatrix}
\]

The determinant of this matrix is $-1.7082274230870981 \times 10^{-9} \approx 0$ (The matrix here differ from original one by omitting from third decimal decimal point).

### 1.4.1 Non-homogeneous Dirichlet Condition

If we want solve the problem
\[
-\Delta u = f \quad \text{in} \ \Omega; \quad u = g \quad \text{on} \ \partial \Omega
\]
We rewrite Example 1 as

5: \texttt{func } f = x \ast y; \texttt{func } g = \sin(\pi x) \ast \cos(\pi y);
6: \texttt{problem \ Poisson}(u,v,\texttt{solver}=\texttt{LU}) =
7: \texttt{int2d}(\texttt{Th})(\texttt{dx}(u) \ast \texttt{dx}(v) + \texttt{dy}(u) \ast \texttt{dy}(v)) \quad // \quad \text{bilinear part}
8: - \texttt{int2d}(\texttt{Th})( f \ast v) \quad // \quad \text{right hand side}
9: + \texttt{on}(1, u=g) ; \quad // \quad \text{Non-homogeneous \ Dirichlet}

This make the following linear system, for \( i \not\in I_\Omega \),

\[
Eu_i + \sum_{j \neq i} A_{ij} u_j = b_i + Eg(q^i)
\]

which means that

\[
u_i = g(q^i) + (b_i - \sum_{j \neq i} A_{ij} u_j) \times E^{-1} \simeq g(q^i) + O(1/E)
\]

**Note 4** To solve non-homogeneous Dirichlet, we rewrite Example 2 as

4: \texttt{Vh } u,v,f,g,bc; \quad g = \sin(\pi x) \ast \cos(\pi y);
5: \texttt{varf } a(u,v) = \texttt{int2d}(\texttt{Th})( \texttt{dx}(u) \ast \texttt{dx}(v) + \texttt{dy}(u) \ast \texttt{dy}(v))
6: + \texttt{on}(C, u=1) ; \quad \texttt{see } (1.7)
10: \texttt{matrix } A=\texttt{a(Vh,Vh)}; \quad \texttt{bc[]}=\texttt{a(0,Vh)};
12: F[]=\texttt{B} \ast \texttt{f[]}; \quad F[] += \texttt{bc[]} \ast \texttt{g[]};

Here “\texttt{bc[]} = \texttt{a(0,Vh)}” create the vector \( [bc_1, bc_2, \cdots, bc_M] \), \( bc_i = 0 \) if \( i \in I_\Omega \) and \( bc_i = E (= 10^{30}) \) if \( i \not\in I_\Omega \). If the finite approximation of \( g \) is \( g \approx g_1 \phi_1 + \cdots + g_M \phi_M \)

\[
bc[] \ast g[] = \sum_{j=1}^{M} bc_j g_j \quad (1.11)
\]

### 1.4.2 Matrix Operations

The multiplicative operators \( \ast, /, \text{ and } \% \) group left to right.

- ‘’ is unary right transposition of array, matrix
- ‘.’ \ast is the term to term multiply operator.
- ‘.’ / is the term to term divide operator.

there are some compound operator:

- \( ^{-1} \) is for solving the linear system (example: \( \texttt{b} = \texttt{A}^{-1} \texttt{x} \))
- ‘’ \ast is the compound of transposition and matrix product, so it is the dot product (example real \texttt{DotProduct} = a’ \ast b)
Example 3

```plaintext
mesh Th = square(2,1);
fespace Vh(Th,P1);
Vh f,g;
f = x*y;
g = sin(pi*x);
Vh<complex> ff,gg; // a complex valued finite element function
ff = x*(y+1i);
ff = exp(pi*x*1i);
varf mat(u,v) = int2d(Th)(1*dx(u)*dx(v)+2*dx(u)*dy(v)+3*dy(u)*dx(v)+4*dy(u)*dy(v))
+ on(1,2,3,4,u=1);
varf mati(u,v) = int2d(Th)(1*dx(u)*dx(v)+2i*dx(u)*dy(v)+3*dy(u)*dx(v)+4*dy(u)*dy(v))
+ on(1,2,3,4,u=1);
matrix A = mat(Vh,Vh); // a complex sparce matrix
Vh m0; m0[] = A * f[];
Vh m01; m01[] = A' * f[];
Vh m1; m1[] = f[].*g[];
Vh m2; m2[] = f[]./g[];
cout << "f = " << f[] << endl;
cout << "g = " << g[] << endl;
cout << "A = " << A << endl;
cout << "m0 = " << m0[] << endl;
cout << "m01 = " << m01[] << endl;
cout << "m1 = " << m1[] << endl;
cout << "m2 = " << m2[] << endl;
cout << "dot Product = " << f[]'*g[] << endl;
cout << "hermitian Product = " << ff[]'*gg[] << endl;
```

This produce the following:

\[
A = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
10^{30} & 10.5 & 0 & 30. & 4 - 2.5 & 0 \\
2 & 10^{30} & 0.5 & 0 & 0.5 & -2.5 \\
3 & 0 & 0. & 10^{30} & 0 & 0.5 \\
4 & 0.5 & 0 & 0 & 10^{30} & 0. \\
5 & -2.5 & 0.5 & 0.5 & 10^{30} & 0. \\
6 & 0 & -2.5 & 0. & 0.5 & 10^{30}
\end{bmatrix}
\]

\[
\{v\} = f[] = (0 \ 0 \ 0 \ 0.5 \ 1)^T
\]

\[
\{w\} = g[] = (0 \ 1 \ 1.2 \times 10^{-16} \ 0 \ 1 \ 1.2 \times 10^{-16})^T
\]

\[
A * f[] = (\begin{bmatrix}
-1.25 & -2.25 & 0.5 & 0 & 5 \times 10^{29} & 10^{30}
\end{bmatrix})^T (= A\{v\})
\]

\[
A' * f[] = (\begin{bmatrix}
-1.25 & -2.25 & 0 & 0.25 & 5 \times 10^{29} & 10^{30}
\end{bmatrix})^T (= A^T\{v\})
\]

\[
f[].*g[] = (0 \ 0 \ 0 \ 0.5 \ 1.2 \times 10^{-16})^T = (v_1 w_1 \ \cdots \ v_M w_M)^T
\]

\[
f[]./g[] = (nan \ 0 \ 0 \ -nan \ 0.5 \ 8.1 \times 10^{15})^T = (v_1/w_1 \ \cdots \ v_M/w_M)^T
\]

\[
f[]' * g[] = 0.5 (= \{v\}\{w\} = \{v\} \cdot \{w\})
\]
1.5 Modeling–Edit–Run–Visualize–Revise

freefem++ provide many examples and its documentation, so you can easily calculate mathematical models by FEM (finite element method) and study them. Explanations for these examples are given in this book. If you are a beginner of FEM, you start from Quick Tour of freefem++. The numerical simulation of scientific problem will be done as follows.

Modeling: Make a mathematical model describing scientific problems. Mathematical modeling is a deep and fruitful one, with many important implications for scientific problems (refer to Chapter 7).

Programming: Translate the mathematical model to freefem++ source code, which is easy because freefem++ includes many clever techniques in FEM with mathematical writing.

Run: Next step is to run it to see if it works. If we provisionally give the name of the source code to “something.edp”, we can execute it by the typing

```
% freefem++ something.edp
```

An important part in programming is to keep aware of collections of programs that are available, and this manual contains many examples you can use freely. So we hope you to run these examples and their representing mathematical models, which are contained in the package in freefem++.

Visualization: The numerical calculation by FEM make huge data, so the easy way to check the obtained result is their visualization. freefem++ can display the mesh and the contour lines of obtained functions. If you want to use these visualization after execution, you add the filename of PostScript to the commands “plot” (see Section 5.1).

Debugging: If the boolean value of “wait” is true (default is ‘false’), then freefem++ will stop at the information in visual form. Write the following, execute it and make a change the line “wait=true” to “wait=false”.

```
bool wait = true; // set "true" if you want see each plotting
mesh Th = square(10,10,[-1+2*x,-1+2*y]); // ]−1,1[^2]
plot(Th); // plot the mesh
fespace Vh(Th,P2);
Vh f = sin(pi*x)*cos(pi*y);
plot(f,wait=wait);
Vh g = sin(pi*x + cos(pi*y));
plot(g,wait=wait); // plot the function f
```
If there is a fatal error in your source code, freefem++ will end and cause an error message to appear. In MS-Windows, freefem++ will open the message file by notepad. For example, if you forget parenthesis as in

```plaintext
mesh Th = square(10,10;
plot(Th);
```

then you will get the following message from freefem++,

```plaintext
mesh Th = square(10,10;
  Error line number 0, in file xxxxxx.edp, before token ;
parse error
Compile error : parse error
  line number :0, ;
at exec line 0
error Compile error : parse error
  line number :0, ;
```

If you use the same symbol twice as in

```plaintext
real aaa =1;
real aaa;
```

then you will get the message

```plaintext
real aaa =1;
  l : real aaa; The identifier aaa exist
```

Notice that the line number start from 0. If you find that the program isn’t doing what you want it to do, then you check the line number and try to figure out what’s wrong. We give two techniques; One is trace by plot for meshes and (FE-)functions with wait=true, and by cout for scalar, vectors and matrices. Another is to comment out by “//”. If you find a doubtful line in your source code, you comment out as follows,

```plaintext
real aaa =1;
  // real aaa;
```

### 1.6 Installation

There are binary packages available for Microsoft Windows and Apple Mac OS. For all other platforms, freefem++ must be compiled and installed from the source archive. This archive is available from:

[http://www.ann.jussieu.fr/~hecht/ftp/freefem/freefem++.tgz](http://www.ann.jussieu.fr/~hecht/ftp/freefem/freefem++.tgz)

To extract files from the compressed archive freefem++.tgz into a directory called freefem++-X.XX (where X.XX is the version number) enter the following commands in a shell window:

```plaintext
tar zxvf freefem++.tgz
cd freefem++-X.XX
```

To compile and install freefem++ , just follow the INSTALL and README files. The following programs are produced, depending on the system you are running (Linux, Windows, MacOS):

```plaintext
```
1. **FreeFem++**, standard version, with a graphical interface based on X11, Win32 or MacOS

2. **FreeFem++-nw**, postscript plot output only (batch version, no windows)

3. **FreeFem++-mpi**, parallel version, postscript output only

4. **FreeFem++-glx**, graphics using OpenGL and X11

5. **FreeFem++-cs**, integrated development environment (please see chapter “Graphical User Interface” for more details).

6. `/Applications/FreeFem++.app`, Drag and Drop CoCoa MacOS Application

7. **FreeFem++-CoCoa**, MacOS Shell script for MacOS OpenGL version (MacOS 10.2 or better) (note: it uses /Applications/FreeFem++.app)

As an installation test, go into the directory examples++-tutorial and run **freefem++** on the example script `LaplaceP1.edp` with the command:

```
FreeFem++ LaplaceP1.edp
```
Chapter 2

Syntax

2.1 Data Types

Basically freefem++ is a compiler, the language is typed, polymorphic and reentrant. Every variable must be typed, declared in a statement; each statement separated from the next by a semicolon ‘;’. The language allows the manipulation of basic types integers (int), reals (real), strings (string), arrays (example: real[int]), bidimensional (2D) finite element meshes (mesh), 2D finite element spaces (fespace), definition of functions (func), arrays of finite element functions (func[basic_type]), linear and bilinear operators, sparse matrices, vectors, etc. For instance

```cpp
int i,n=20; // i,n are integer.
real[int] xx(n),yy(n); // two array of size n
for (i=0;i<=20;i++) // which can be used in statements such as
{ xx[i]= cos(i*pi/10); yy[i]= sin(i*pi/10); }
```

The life of a variable is the current block {...}, except the fespace variable, and the in variables local to a block are destroyed at the end of the block as follows.

Example 4

```cpp
real r= 0.01;
mesh Th=square(10,10); // unit square mesh
fespace Vh(Th,P1); // P1 lagrange finite element space
Vh u = x+ exp(y);
func f = z * x + r * log(y);
plot(u,wait=true);
{
    real r = 2; // not the same r
    fespace Vh(Th,P1); // error because Vh is a global name
}
```

The type declarations are compulsory in freefem++ because it is easy to make bugs in a language with many types. The variable name is just an alphanumeric string, the underscore character “_” is not allowed, because it will be used as an operator in the future.
2.2 List of major types

bool is used for logical expression and flow-control.

int declare an integer.

string declare the variable to store a text enclosed within double quotes, such as:
"This is a string in double quotes."

real declare the variable to store a number such as “12.345”.

complex Complex numbers, such as 1 + 2i, $i = \sqrt{-1}$.

```c++
complex a = 1i, b = 2 + 3i;
cout << "a + b = " << a + b << endl;
cout << "a - b = " << a + b << endl;
cout << "a * b = " << a + b << endl;
cout << "a / b = " << a / b << endl;
```

Here’s the output:

```
a + b = (2,4)
a - b = (-2,-2)
a * b = (-3,2)
a / b = (0.230769,0.153846)
```

ofstream make a output file and its functions.

ifstream make a input file and its functions.

real[int ] declare a variable that store multiple real numbers with integer index.

```c++
real[int ] a(5);
cout << "a = " << a << endl;
```

This produces the output:
```
a = 5 :
   1 2 3.33333 4 5
```

real[string ] declare a variable that store multiple real numbers with string index.

string[string ] declare a variable that store multiple strings with string index.

func define a function without argument, if independent variables are $x, y$. For example

```c++
func f=cos(x)+sin(y);
```

Remark the function’s type is given by the expression’s type. The power of functions are given in freefem++ such as $x^1, y^{0.23}$.

mesh create the triangulation, see Section §3.

fespace define a new type of finite element space, see Section Section §4.
problem declare the weak form of a partial differential problem without solving.
solve declare a problem and solve it.
varf define a full variational form.
matrix define a sparse matrix.

2.3 Global Variables

The names $x, y, z, \text{label}, \text{region}, P, N, \text{nu\_triangle}$ are used to link the language to the finite element tools:

- $x$ expresses $x$ coordinate of current point (real value)
- $y$ expresses $y$ coordinate of current point (real value)
- $z$ expresses $z$ coordinate of current point (real value), but is reserved for future use.
- $\text{label}$ show the label number of boundary if the current point is on a boundary, otherwise 0 (int value).
- $\text{region}$ returns the region number of the current point $(x,y)$ (int value).
- $P$ give the current point ($\mathbb{R}^2$ value). By $P.x$, $P.y$, we can get the $x, y$ components of $P$. Also $P.z$ is reserved.
- $N$ give the outward unit normal vector at the current point is on the curve defined by $\text{border}$ ($\mathbb{R}^3$ value). $N.x$ and $N.y$ are $x$ and $y$ components of the normal vector. $N.z$ is reserved.
- $\text{len\_Edge}$ give the length of the current edge
  \[ \text{len\_Edge} = |q^i - q^j| \text{ if the current edge is } [q^i, q^j] \]
- $\text{h\_Triangle}$ give the size of the current triangle
- $\text{nu\_Triangle}$ give the index of the current triangle (integer).
- $\text{nu\_Edge}$ give the index of the current edge in the triangle (integer).
- $\text{n\_Ton\_Edge}$ give the number of adjacent triangle of the current edge (integer).
- $\text{area}$ give the area of the current triangle (real).
- $\text{cout}$ is the standard output device (default is console). On MS-Windows, the standard output is only to console, in this time. $\text{ostream}$
- $\text{cin}$ is the standard input device (default is keyboard). ($\text{istream}$). On MS-Windows, this don’t work.
24

CHAPTER 2. SYNTAX

**endl** give the end of line in the input/output devices.

**true** means “true” in bool value.

**false** means “false” in bool value.

**pi** is the real approximation value of \( \pi \).

Here is how to show all the types, and all the operator and functions.

```cpp
dumpTable(cout);
```

To execute a system command in the string (not implemented on Carbon MacOs)

```cpp
exec("shell command");
```

On MS-Windows, we need the full path. For example, if there is the command “ls.exe” in the subdirectory “c:\cygwin\bin\”, then we must write

```cpp
exec("c:\cygwin\bin\ls.exe");
```

## 2.4 Arithmetic

In integers, +, −, ∗ express the usual arithmetic summation (plus), subtraction (minus) and multiplication (times), respectively. The operators / and % yield the quotient and the remainder from the division of the first expression by the second. If the second number of / or % is zero the behavior is undefined. The **maximum** or **minimum** of two integers \( a, b \) are obtained by max \((a, b)\) or min \((a, b)\). The power \( a^b \) of two integers \( a, b \) is calculated by writing \( a^b \).

**Example 5** Calculations with the integers

```cpp
int a = 12, b = 5;
cout <<"plus, minus of " <<a<<" and " <<b<<" are " <<a+b<<endl;
cout <<"multiplication, quotient of them are " <<a*b<<"," <<a/b<<endl;
cout <<"remainder from division of " <<a<<" by " <<b<<" is " <<a%b<<endl;
cout <<"the minus of " <<a<<" is " <<-a<< endl;
cout <<a<<" plus -" <<b<<" need bracket:" <<a<<"+(-" <<b<<")"=" <<a+(b)<<endl;
cout <<"max and min of " <<a<<" and " <<b<<" is " <<max(a,b)<<"," <<min(a,b)<<endl;
cout <<b<<"th power of " <<a<<" is " <<a^b<< endl;
b=0;
cout <<a"/0"<<" is " a/b<< endl;
cout <<a"%0"<<" is " a%b<< endl;
```

produce the following result:

- plus, minus of 12 and 5 are 17, 7
- multiplication, quotient of them are 60, 2
- remainder from division of 12 by 5 is 2
- the minus of 12 is -12
- 12 plus -5 need bracket :12+(-5)=7
- max and min of 12 and 5 is 12,5
5th power of 12 is 248832
12/0 : long long long
Fatal error : ExecError Div by 0 at exec line 9
Exec error : exit

By the relation \( \text{integer} \subset \text{real} \), the operators “+,” “−,” “∗,” “/,” “%” and “\text{max}, \text{min}, \text{−}” are also applicable in real-type. However, “%” calculates the remainder of the integral parts of two real numbers.

The following example similar to Example 5

```cpp
real a=sqrt(2.), b = pi;
cout <<"plus, minus of "<<a<<" and "<<pi<<" are "<< a+b <<", "<< a-b << endl;
cout <<"multiplication, quotient of them are "<<a*b<<", "<<a/b<< endl;
cout <<"remainder from division of "<<a<<" by "<<b<<" is "<< a%b << endl;
cout <<"the minus of "<<a<<" is "<< -a << endl;
cout <<a<<" plus "<<b<<" need bracket :"<<a<<"+("<<b<<")="<<a + (-b) << endl;
```
gives the following output:

plus, minus of 1.41421 and 3.14159 are 4.55581, -1.72738
multiplication, quotient of them are 4.44288, 0.450158
remainder from division of 1.41421 by 3.14159 is 1
the minus of 1.41421 is -1.41421
1.41421 plus -3.14159 need bracket :1.41421+(-3.14159)=-1.72738

By the relation

\[ \text{bool} \subset \text{int} \subset \text{real} \subset \text{complex}, \]

the operators “+,” “−,” “∗,” “/” and “−” are also applicable in complex-type, but “%,” “\text{max}, \text{min}” fall into disuse. Complex number such as \(5+9i, i=\sqrt{-1}, \) can be a little tricky. For real variables \(a=2.45, \ b=5.33\), we must write the complex numbers \(a+b*1i\) and \(a+\text{sqrt}(2.0)*1i\) as

```cpp
complex z1 = a+b*1i, z2=a+sqrt(2.0)*1i;
```
The imaginary and real parts of complex number \(z\) is obtained by \text{imag} and \text{real}. The conjugate of \(a+bi\) \((a, b \text{ are real})\) is defined by \(a−bi\), which is denoted by \text{conj}(a+b*1i) in \text{freefem++}.

The complex number \(z = a+ib\) is considered as the pair \((a, b)\) of real numbers \(a, b\). Now we draw the point \((a, b)\) in the plane (Cartesian rectangular system of axes) and mark on the \(x\)-axis the real numbers in the usual way, on the \(y\)-axis the imaginary numbers with \(i\) as unit. By changing Cartesian coordinate \((a, b)\) to the polar coordinate \((r, φ)\), the complex number \(z\) has another expression \(z = r(\cos φ + i \sin φ)\), \(r = \sqrt{a^2 + b^2}\) and \(φ = \tan^{-1}(b/a)\); \(r\) is called the absolute value and \(φ\) the argument of \(z\). In the following example, we shall show them using \text{freefem++} programming, and \text{de Moivre's formula} \(z^n = r^n(\cos nφ + i \sin nφ)\).

**Example 6**

```cpp
real a=2.45, b=5.33;
complex z1=a+b*1i, z2 = a+sqrt(2.0)*1i;
```


```cpp
func string pc(complex z) // printout complex to (real)+i(imaginary)
{
    string r = "("+real(z);
    if (imag(z)>=0) r = r+"+";
    return r+imag(z)+"i");
}

// printout complex to |z|*(cos(arg(z))+i*sin(arg(z)))
func string toPolar(complex z)
{
    return abs(z)*" *(cos("+arg(z)+")+i * sin("+arg(z)+"))";
}
cout <<"Standard output of the complex "<<pc(z1)<<" is the pair "
<<z1<<endl;
cout <<"Plus, minus of "<<pc(z1)<<" and "<<pc(z2)<<" are "<<pc(z1+z2)
<<", "<<pc(z1-z2)<<" endl;
cout <<"Multiplication, quotient of them are "<<pc(z1*z2)<<", "
<<pc(z1/z2)<<" endl;
cout <<"Real/imaginary part of "<<pc(z1)<<" is "<<real(z1)<<", "
<<imag(z1)<<" endl;
cout <<"Absolute of "<<pc(z1)<<" is "<<abs(z1)<<" endl;
cout <<pc(z2)<<" = "<<toPolar(z2)<<" endl;
cout <<" and polar("<<abs(z2)<<", "<<arg(z2)<<") = "
<<pc(polar(abs(z2),arg(z2)))<<" endl;
cout <<"de Moivre's formula: "<<pc(z2)"^3 = "<<toPolar(z2^3)<<" endl;
cout <<"conjugate of "<<pc(z2)<<" is "<<pc conj(z2)<<" endl;
cout <<pc(z1)<<""<<pc(z2)<<" is "<<pc(z1*z2)<<" endl;
```

Here's the output from Example 6

Standard output of the complex (2.45+5.33i) is the pair
(2.45,5.33)
Plus, minus of (2.45+5.33i) and (2.45+1.41421i) are
(4.9+6.74421i), (0+3.91579i)
Multiplication, quotient of them are (-1.53526+16.5233i),
(1.692+1.19883i)
Real/imaginary part of (2.45+5.33i) is 2.45, 5.33
Absolute of (2.45+5.33i) is 5.86612
(2.45+1.41421i) = 2.82887*(cos(0.523509)+i*sin(0.523509))
and polar(2.82887,0.523509) = (2.45+1.41421i)
de Moivre’s formula: (2.45+1.41421i)^3
= 22.638*(cos(1.57053)+i*sin(1.57053))
conjugate of (2.45+1.41421i) is (2.45-1.41421i)
(2.45+5.33i)^(2.45+1.41421i) is (8.37072-12.7078i)

2.5 One Variable Functions

Fundamental functions are built into freefem++. The power function \( x^\alpha \) (\( x^\alpha \)); the exponent function \( \exp(x) = e^x \); the logarithmic function \( \log(x) = \ln x \) or \( \log_10(x) \) (= \( \log_{10} x \)); the trigonometric functions \( \sin(x) \), \( \cos(x) \), \( \tan(x) \) depending on angles measured by \( \text{radian} \); the inverse of \( \sin x \), \( \cos x \), \( \tan x \) called circular function or inverse
2.5. ONE VARIABLE FUNCTIONS

trigonometric function \( \text{asin}(x) (= \arcsin x), \text{acos}(x) (= \arccos x), \text{atan}(x) (= \arctan x) \);
the hyperbolic function,

\[
\sinh x = \left( e^x - e^{-x} \right)/2, \quad \cosh x = \left( e^x + e^{-x} \right)/2.
\]

and \( \tanh x = \sinh x/\cosh x \) written by \( \sinh(x), \cosh(x), \text{asin}(x) \) and \( \text{acosh}(x) \).

\[
\sinh^{-1} x = \ln \left[ x + \sqrt{x^2 + 1} \right], \quad \cosh^{-1} x = \ln \left[ x + \sqrt{x^2 - 1} \right].
\]

Elementary Functions is the class of functions consisting of the functions in this section (polynomials, exponential, logarithmic, trigonometric, circular) and the functions obtained from those listed by the four arithmetic operations

\[
f(x) + g(x), \quad f(x) - g(x), \quad f(x)g(x), \quad f(x)/g(x)
\]

and by superposition \( f(g(x)) \), in which four arithmetic operations and superpositions are permitted finitely many times. In freefem++ , we can create all elementary functions. The derivative of an elementary function is also elementary. However, the indefinite integral of an elementary function cannot always be expressed in terms of elementary functions.

Example 7 The following give the example to make the boundary using elementary functions. Cardioid

```c
real b = 1.;
real a = b;
func real phix(real t) {
  return (a+b)*cos(t)-b*cos(t*(a+b)/b);
}
func real phiy(real t) {
  return (a+b)*sin(t)-b*sin(t*(a+b)/b);
}
border C(t=0,2*pi) { x=phix(t); y=phiy(t); }
mesh Th = buildmesh(C(50));
```

Taking the principal value, we can define \( \log z \) for \( z \neq 0 \) by

\[
\log z = \ln |z| + \arg z.
\]

Using freefem++ , we calculated \( \exp(1+4i), \sin(\pi+1i), \cos(\pi/2-1i) \) and \( \log(1+2i) \), we then have

\[
-1.77679 - 2.0572i, \quad 1.8896710^{-16} - 1.1752i, \\
9.4483310^{-17} + 1.1752i, \quad 0.804719 + 1.10715i.
\]
CHAPTER 2. SYNTAX

2.6 Two Variable Functions

2.6.1 Formula

The general form of real functions with two independent variables $x$, $y$ is usually written as $z = f(x,y)$. In freefem++, $x$ and $y$ are reserved word in Section 2.3. When two independent variables are $x$ and $y$, we can define a function without argument, for example

```cpp
func f = cos(x) + sin(y);
```

Remark the function’s type is given by the expression’s type. The power of functions are given in freefem++ such as $x^{1.2}$, $y^{0.23}$. In func, we can write an elementary function as follows

```cpp
func f = sin(x) * cos(y);
func g = (x^2 + 3*y^2) * exp(1 - x^2 - y^2);
func h = max(-0.5, 0.1 * log(f^2 + g^2));
```

Complex valued function create functions with 2 variables $x$, $y$ as follows,

```cpp
mesh Th = square(20, 20, [-pi + 2*pi*x, -pi + 2*pi*y]);
func f = x^2 * (1+y)^3 + y^2;
```

2.6.2 FE-function

Arithmetic built-in functions are able to construct a new function by the four arithmetic operations and superposition of them (see elementary functions), which are called formulas to distinguish from FE-functions. We can add new formulas easily, if we want. Here, FE-function is an element of finite element space (real or complex) (see Section Section 4). Or to put it another way: formulas are the mathematical expressions combining its numerical analogs, but it is independent of meshes (triangulations).

Also, in freefem++, we can give an arbitrary symbol to FE-function combining numerical calculation by FEM. The projection of a formula $f$ to FE-space is done as in

```cpp
func f = x^2 * (1+y)^3 + y^2;
```

We call also by two variable elementary function functions obtained from elementary functions $f(x)$ or $g(y)$ by the four arithmetic operations and by superposition in finite times.

Note 6 The command plot is valid only for real FE-functions.
Complex valued function create functions with 2 variables \( x, y \) as follows,

```cpp
mesh Th=square(20,20,[-pi+2*pi*x,-pi+2*pi*y]); // ]-\pi,\pi]^2
fespace Vh(Th,P2);
func z=x+y*1i; // z = x + iy
func f=imag(sqrt(z)); // f = \Im \sqrt{z}
func g=abs( sin(z/10)*exp(z^2/10) ); // g = |\sin(z/10)\exp(z^2/10)|
Vh fh = f; plot(fh);    // Fig. 2.1 isovalue of f
Vh gh = g; plot(gh);    // Fig. 2.2 isovalue of g
```

Figure 2.1: \( \sqrt{z} \) has branch

Figure 2.2: \(|\sin(z/10)\exp(z^2/10)|\)

### 2.7 Array

An array stores multiple objects, and there are 2 kinds of arrays: The first is the vector that is arrays with integer indices and arrays with string indices.

In the first case, the size of this array must be know at the execution time, and the implementation is done with the `KN<>` class so all the vector operator of `KN<>` are implemented.

The sample

```cpp
real [int] tab(10), tab1(10);    // 2 array of 10 real
real [int] tab2;                // bug array with no size
tab = 1.03;                     // set all the array to 1.03
Tab[1]=2.15;
cout << tab[1] << " " << tab[9] << " size of tab = " << tab.n << " " << tab.min << " " << tab.max << " " <<
endl;
```

produce the output

```
2.15 1.03 size of tab = 10 1.03 2.15
```

It is also possible to make an array of FE function, with the same syntax, and we can treat them as vector valued function if we need them.

**Example 8** In the following example, Poisson’s equation is solved under 3 different given functions \( f = 1, \sin(\pi x)\cos(\pi y), |x-1||y-1| \), whose solutions are stored in an array of FE
function.

```
mesh Th = square(20, 20, [2*x, 2*y]);
fespace Vh(Th, P1);
Vh u, v, f;
problem Poisson(u, v) =
    int2d(Th)( dx(u) * dx(v) + dy(u) * dy(v) )
    + int2d(Th)( -f * v ) + on(1, 2, 3, 4, u=0);
Vh[int] uu(3);  // an array of FE function
f=1;  // problem1
Poisson; uu[0] = u;
f=sin(pi*x) * cos(pi*y);  // problem2
Poisson; uu[1] = u;
f=abs(x-1) * abs(y-1);  // problem3
Poisson; uu[2] = u;
for (int i=0; i<3; i++)  // plots all solutions
    plot(uu[i], wait=true);
```

For the second case, it is just a map of the STL\[?] so no vector operation except the selection of an item is allowed.

The transpose operator is ' like MathLab or SciLab, so the way to compute the dot product of two array a,b is `real ab = a’*b`.

```
int i;
real [int] tab(10), tabl(10);  // 2 array of 10 real
    real [int] tab2;  // bug array with no size
    tab = 1;  // set all the array to 1
    tab[1]=2;
    cout << tab[1] << " " << tab[9] << " size of tab = "
        << tab.n << " " << tab.min << " " << tab.max << " " << endl;
    tabl=tab;
    tab=tab+tabl;
    tab=2*tab+tabl*5;
    tabl=2*tab-tabl*5;
    tab+=tab;
    cout << " dot product " << tab’*tab << endl;  // tabtab
    cout << tab << endl;
    real[string] map;  // a dynamique array
    for (i=0; i<10; i=i+1)
    {
        tab[i] = i*i;
        cout << i << " " << tab[i] << "\n";
    };
    map["1"] = 2.0;
    map[2] = 3.0;  // 2 is automatically cast to the string "2"
    cout << " map["1"] = " << map["1"] << "; " << endl;
```

\[1\] Standard template Library, now part of standard C++
2.8 Loops

The for and while loops are implemented with break and continue keywords. In for-loop, there are three parameters; the INITIALIZATION of a control variable, the CONDITION to continue, the CHANGE of the control variable. While CONDITION is true, for-loop continue.

\begin{verbatim}
for (INITIALIZATION; CONDITION; CHANGE)
{ BLOCK of calculations }
\end{verbatim}

The sum from 1 to 10 is calculated by (the result is in sum),

\begin{verbatim}
int sum=0;
for (int i=1; i<=10; i++)
    sum += i;
\end{verbatim}

The while-loop

\begin{verbatim}
while (CONDITION) {
    BLOCK of calculations or change of control variables
}
\end{verbatim}

is executed repeatedly until CONDITION become false. The sum from 1 to 10 is also written by while as follows,

\begin{verbatim}
int i=1, sum=0;
while (i<=10) {
    sum += i; i++;
}
\end{verbatim}

We can exit from a loop in midstream by break. The continue statement will pass the part from continue to the end of the loop.

Example 9

\begin{verbatim}
for (int i=0;i<10;i=i+1)
    cout << i << "\n";
real eps=1;
while (eps>1e-5)
{ eps = eps/2;
    if( i++ <100) break;
    cout << eps << endl;}
for (int j=0; j<20; j++) {
    if (j<10) continue;
    cout << "j = " << j << endl;
}\end{verbatim}

2.9 Input/Output

The syntax of input/output statements is similar to C++ syntax. It uses \texttt{cout, cin, endl, <<,>>}. To write to (resp. read from) a file, declare a new variable ofstream ofile("filename"); or ofstream ofile("filename", append); (resp. ifstream ifile("filename");
and use ofstream (resp. ifstream) as cout (resp. cin). The word append in ofstream ofstream("filename", append); means opening a file in append mode.

Note 7 The file is closed at the exit of the enclosing block.

Example 10

```cpp
int i;
cout << " std-out" << endl;
cout << " enter i= ? ";
cin >> i ;
{
    ofstream f("toto.txt");
    f << i << "coucou\n";
} // close the file f because the variable f is delete

{ ifstream f("toto.txt");
    f >> i;
}
{
    ofstream f("toto.txt", append);
    // to append to the existing file "toto.txt"
    f << i << "coucou\n";
} // close the file f because the variable f is delete

cout << i << endl;
```
Chapter 3

Mesh Generation

3.1 Commands for Mesh Generation

In Step1 in Section 1.2, the keywords **border**, **buildmesh** are explained. All the examples in this section come from the files `mesh.edp` and `tablefunction.edp`.

3.1.1 Square

For easy and simple testing, there is the command **“square”**. The following

```javascript
mesh Th = square(4, 5);
```

generate a $4 \times 5$ grid in the unit square $[0, 1]^2$ whose labels are shown in Fig. 3.1. If you want

![Boundary labels of the mesh by square(10, 10)](image)

Figure 3.1: Boundary labels of the mesh by `square(10, 10)`

constructs a $n \times m$ grid in the rectangle $[x_0, x_1] \times [y_0, y_1]$, you can write

```javascript
real x0=1.2, x1=1.8;
real y0=0, y1=1;
int n=5, m=20;
mesh Th=square(n,m, [x0+(x1-x0)*x, y0+(y1-y0)*y]);
```
3.1.2 Border

A domain is defined as being on the left (resp right) of its parameterized boundary

\[ \Gamma_j = \{(x, y) \mid x = \varphi_x(t), y = \varphi_y(t), a_j \leq t \leq b_j\} \]

We can easily check the orientation by drawing the curve \( t \mapsto (\varphi_x(t), \varphi_y(t)), t_0 \leq t \leq t_1 \). If the figure become like to Fig. 3.2, then the domain lie on the shaded area, otherwise it lie on opposite side (see also the examples enclosed with the box). The boundaries \( \Gamma_j \) can only intersect at their end points.

\[ G_j(t) = t_0 \leq t \leq t_1 \]

**Figure 3.2:** Orientation of the boundary defined by \((\varphi_x(t), \varphi_y(t))\)

The general expression of the triangulation is

```
mesh Mesh_Name = buildmesh(\Gamma_1(m_1) + \cdots + \Gamma_J(m_J));
```

where \( m_j \) are numbers of marked points on \( \Gamma_j \), \( \Gamma = \bigcup_{j=1}^J \Gamma_j \). We can change the orientation of boundaries by changing the sign of \( m_j \). The following example shows how to change the orientation. The example generates the unit disk with a small circular hole, and assign “1” to the unit disk (“2” to the circle inside). The boundary label must be non-zero, however we can omit the label if we want use only the symbol.

```latex
1: border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
2: border b(t=0,2*pi){ x=0.3+0.3*cos(t); y=0.3*sin(t);label=2;}
3: plot(a(50)+b(+30)) ;  // to see a plot of the border mesh
4: mesh Thwithouthole = buildmesh(a(50)+b(+30));
5: mesh Thwithhole = buildmesh(a(50)+b(-30));
6: plot(Thwithouthole,wait=1,ps="Thwithouthole.eps");  // figure 3.3
7: plot(Thwithhole,wait=1,ps="Thwithhole.eps");  // figure 3.4
```

**Note 8** You must notice that the orientation is changed by “b(-30)” in 5th line. In 7th line, `ps="fileName"` is used to generate a postscript file identification that is shown on screen.
3.1.3 Data Structure of Mesh and Reading/Writing a Mesh

Some user asked us that they want use the triangulation made from other tools or hand-made mesh. The example

```
border C(t=0,2*pi) { x=cos(t); y=sin(t); }
mesh Th = buildmesh(C(10));
savemesh("mesh_sample.msh");
```

make the mesh as in Fig. 3.5. The informations about $\text{Th}$ are save in the file “mesh_sample.msh”. We can read from Fig. 3.5 and “mesh_sample.msh” as in Table 3.1 where $n_v$ is the number of vertices, $n_t$ number of triangles and $n_e$ the number of edges on boundary. For each vertex $q^i$, $i = 1, \cdots , n_v$, we denote by $(q_x^i, q_y^i)$ the $x$-coordinate and $y$-coordinate.

Each triangle $T_k$, $k = 1, \cdots , 10$ have three vertices $q^{k_1}$, $q^{k_2}$, $q^{k_3}$ that are oriented in counter-clockwise. The boundary consists of 10 lines $L_i$, $i = 1, \cdots , 10$ whose tips are $q^{i_1}$, $q^{i_2}$.

<table>
<thead>
<tr>
<th>Contents of file</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 16 10</td>
<td>$n_v$ $n_t$ $n_e$</td>
</tr>
<tr>
<td>-0.309016994375 0.951056516295 1</td>
<td>$q_x^1$ $q_y^1$ boundary label=1</td>
</tr>
<tr>
<td>0.309016994375 0.951056516295 1</td>
<td>$q_x^2$ $q_y^2$ boundary label=1</td>
</tr>
<tr>
<td>$\cdots \cdots$</td>
<td>$q_x^{14}$ $q_y^{14}$ boundary label=1</td>
</tr>
<tr>
<td>-0.309016994375 -0.951056516295 1</td>
<td></td>
</tr>
<tr>
<td>9 12 10 0</td>
<td>$1_1$ $1_2$ $1_3$ region label=0</td>
</tr>
<tr>
<td>5 9 6 0</td>
<td>$2_1$ $2_2$ $2_3$ region label=0</td>
</tr>
<tr>
<td>$\cdots$</td>
<td>$16_1$ $16_2$ $16_3$ region label=0</td>
</tr>
<tr>
<td>9 10 6 0</td>
<td></td>
</tr>
<tr>
<td>6 5 1</td>
<td>$1_1$ $1_2$ boundary label=1</td>
</tr>
<tr>
<td>5 2 1</td>
<td>$2_1$ $2_2$ boundary label=1</td>
</tr>
<tr>
<td>$\cdots$</td>
<td></td>
</tr>
<tr>
<td>10 6 1</td>
<td>$10_1$ $10_2$ boundary label=1</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: The structure of “mesh_sample.msh”
In the left figure, we have the following.

\[ n_v = 14, \quad n_t = 16, \quad n_s = 10 \]

\[ q^1 = (-0.300016994375, 0.951056516295) \]
\[ \vdots \]
\[ q^{14} = (-0.300016994375, -0.951056516295) \]

The vertices of \( T_1 \) are \( q^0, q^{12}, q^{10} \).
\[ \vdots \]

The vertices of \( T_{16} \) are \( q^0, q^{10}, q^6 \).
\[ \vdots \]

The edge of 1st side \( L_1 \) are \( q^6, q^5 \).
\[ \vdots \]

The edge of 10th side \( L_{10} \) are \( q^{10}, q^6 \).

There are many mesh file formats available for communication with other tools such as emc2, modulef.. (see Section 10), The extension of a file gives the chosen type. More details can be found in the article by F. Hecht "bamg : a bidimensionnal anisotropic mesh generator" available from the FreeFem web site.

The following give the example write and read files of generated mesh, Freefem can read and write files. A mesh file can be read back into \texttt{freefem++} but the names of the borders are lost. So these borders have to be referenced by the number which corresponds to their order of appearance in the program, unless this number is forced by the keyword "label".

```cpp
border floor(t=0,1) { x=t; y=0; label=1; };
// the unit square
border right(t=0,1) { x=1; y=t; label=5; };
border ceiling(t=1,0) { x=t; y=1; label=5; };
border left(t=1,0) { x=0; y=t; label=5; };
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.am_fmt"); // "formatted Marrocco" format
savemesh(th,"toto.Th"); // "bamg"-type mesh
savemesh(th,"toto.msh"); // freemf format
savemesh(th,"toto.nopo"); // modulef format see [8]
mesh th2 = readmesh("toto.msh"); // read the mesh
```

The following example explains methods to obtain mesh information.

```cpp
\{
    mesh Th=square(2,2); // get mesh information (version 1.37)
    int nbtriangles=Th.nt; // get data of the mesh
    for (int i=0;i<nbtriangles;i++)
        for (int j=0; j <3; j++)
            cout << i << " " << j << " Th[i][j] = "
```

\[ \text{\texttt{culm}} \]
3.1. COMMANDS FOR MESH GENERATION

```cpp
<< Th[i][j] << " x = " << Th[i][j].x << " , y= " << Th[i][j].y << ", label=" << Th[i][j].label << endl;
```

the output is:

```
0 0 Th[i][j] = 0 x = 0 , y= 0, label=4
0 1 Th[i][j] = 1 x = 0.5 , y= 0, label=1
0 2 Th[i][j] = 4 x = 0.5 , y= 0.5, label=0
1 0 Th[i][j] = 0 x = 0 , y= 0, label=4
1 1 Th[i][j] = 4 x = 0.5 , y= 0.5, label=0
1 2 Th[i][j] = 3 x = 0 , y= 0.5, label=4
```

```
5 2 Th[i][j] = 6 x = 0 , y= 1, label=4
6 0 Th[i][j] = 4 x = 0.5 , y= 0.5, label=0
6 1 Th[i][j] = 5 x = 1 , y= 0.5, label=2
6 2 Th[i][j] = 8 x = 1 , y= 1, label=3
7 0 Th[i][j] = 4 x = 0.5 , y= 0.5, label=0
7 1 Th[i][j] = 8 x = 1 , y= 1, label=3
7 2 Th[i][j] = 7 x = 0.5 , y= 1, label=3
```

Example 11 (Readmesh.edp)  

```cpp
border floor(t=0,1){ x=t; y=0; label=1;} // the unit square
border right(t=0,1){ x=1; y=t; label=5;}
border ceiling(t=1,0){ x=t; y=1; label=5;}
border left(t=1,0){ x=0; y=t; label=5;}
int n=10;
mesh th= buildmesh(floor(n)+right(n)+ceiling(n)+left(n));
savemesh(th,"toto.Th"); // format database db mesh "bamg"
savemesh(th,"toto.am_fmt"); // format "formatted Marrocco"
savemesh(th,"toto.msh"); // format freefem
savemesh(th,"toto.nopo"); // modulef format see [3]
mesh th2 = readmesh("toto.msh");
fespace fem1(th,P1);
fem1 f = sin(x) * cos(y),g;
{
    ofstream file("f.txt");
    file << f[] << endl;
}
// save solution
ifstream file("f.txt");
file >> g[] ;
// read
fespace Vh2(th2,P1);
Vh2 u,v;
plot (g);
```

```cpp
solve pb(u,v) =
    int2d(th) ( u*v - dx(u)*dx(v)-dy(u)*dy(v) )
+ int2d(th) (-g*v)
+ int1d(th,5) ( g*v)
+ on(1,u=0) ;
plot (th2,u);
```
3.1.4 Triangulate

\texttt{freefem++} is able to build a triangulation from a set of points. This triangulation is a Delaunay mesh of the convex hull of the set of points. It can be useful to build a mesh form a table function.

The coordinates of the points and the value of the table function are defined separately with rows of the form: \( x \ y \ f(x,y) \) in a file such as:

\begin{verbatim}
0.51387 0.175741 0.636237
0.308652 0.534534 0.746765
0.947628 0.171736 0.899823
0.702231 0.226431 0.800819
0.494773 0.12472 0.580623
0.0838988 0.389647 0.456045
...............
\end{verbatim}

Figure 3.6: Delaunay mesh of the convex hull of point set in file xyf

The third column of each line is left untouched by the \texttt{triangulate} command. But you can use this third value to define a table function with rows of the form: \( x \ y \ f(x,y) \). The following example shows how to make mesh from the file “xyf” with the format stated just above. The command \texttt{triangulate} command use only use 1st and 2nd rows.

\begin{verbatim}
mesh Thxy=triangulate("xyf"); // build the Delaunay mesh of the convex hull points are defined by the first 2 columns of file xyf
plot(Thxy,ps="Thxyf.ps"); // (see figure 3.6)

fespace Vhxy(Thxy,P1); // create a P1 interpolation
Vhxy fxy;

// reading the 3rd row to define the function

{ ifstream file("xyf");
  real xx,yy;
  for(int i=0;i<fxy.n;i++)
    file >> xx >>yy >> fxy[i][i]; // to read third row only. xx and yy are just skipped
}
plot(fxy,ps="xyf.eps"); // plot the function (see figure 3.7)
\end{verbatim}

Figure 3.7: Isovalue of table function of point set in file xyf
3.2. **BUILD EMPTY MESH**

When you want to define Finite Element space on boundary, we come up with the idea of a mesh with no internal points (call empty mesh). It can be useful when you have a Lagrange multiplier defined on the border.

So the function emptymesh remove all the internal point of a mesh expect if the point is on internal boundary.

```plaintext

{ // new stuff 2004 emptymesh (version 1.40)
  // -- useful to build Multiplicator space
  // build a mesh without internal point
  // with the same boundary
  // -----

  assert(version>=1.40);
  border a(t=0,2*pi){ x=cos(t); y=sin(t); label=1; }
  mesh Th=buildmesh(a(20));
  Th=emptymesh(Th);
  plot(Th,wait=1,ps="emptymesh-1.eps");  // see figure 3.8
}
```

or it is also possible to build a empty mesh of pseudo subregion with emptymesh(Th, ssd) with the set of edges of the mesh Th a edge e is in this set if the two adjacent triangles $e = t1 \cap t2$ and $ssd[T1] \neq ssd[T2]$ where ssd defined the pseudo region numbering of triangle, when they are stored in int[int] array of size the number of triangles.

```plaintext

{ // new stuff 2004 emptymesh (version 1.40)
  // -- useful to build Multiplicator space
  // build a mesh without internal point
  // of pseudo sub domain
  // -----

  assert(version>=1.40);
  mesh Th=square(10,10);
  int[int] ssd(Th.nt);
  for(int i=0;i<ssd.n;i++)
  {
    int iq=i/2;  // because 2 triangle per quad
    int ix=iq%10;
    int iy=iq/10;
    ssd[i]= 1 + (ix>=5) + (iy>=5)*2;
  }
  Th=emptymesh(Th,ssd);  // build emptu with
  // all edge e = T1 \cap T2 and ssd[T1] \neq ssd[T2]
  plot(Th,wait=1,ps="emptymesh-2.eps");  // see figure 3.9
  savemesh(Th,"emptymesh-2.msh");
}
```

3.3. **Remeshing**

3.3.1. **Movemesh**

After solving the elasticity, we want watch the deformation $\Omega \mapsto \Phi(\Omega)$, $\Phi(x,y) = (\Phi_1(x,y), \Phi_2(x,y))$ of shape. In free boundary value problems, the shape of domain will vary.
CHAPTER 3. MESH GENERATION

Figure 3.8: The empty mesh with boundary Figure 3.9: An empty mesh defined from a pseudo region numbering of triangle

If $\Omega$ is triangulated already — dubbed $T_h(\Omega)$, then we want $\Phi(\Omega)$ is also triangulated automatically. This want is satisfied by

$$\text{mesh } Th = \text{movemesh}(Th, [\Phi_1, \Phi_2]);$$

where $\Phi = (\Phi_1, \Phi_2)$ and $\Phi_i$, $i = 1, 2$ are functions. Sometimes the moved mesh is invalid because some triangle becomes reversed (with a negative area). This is why we check the minimum triangle area in the transformed mesh with checkmovemesh before any real transformation.

**Example 12** $\Phi_1(x, y) = x + k \cdot \sin(y \cdot \pi) / 10$, $\Phi_2(x, y) = y + k \cdot \cos(y \cdot \pi) / 10$ for a big number $k > 1$.

```
verbosity = 4;
border a(t=0,1){x=t;y=0;label=1;};
border b(t=0,0.5){x=1;y=t;label=1;};
border c(t=0,0.5){x=1-t;y=0.5;label=1;};
border d(t=0.5,1){x=0.5;y=t;label=1;};
border e(t=0.5,1){x=1-t;y=1;label=1;};
border f(t=0,1){x=0;y=1-t;label=1;};
func uu = sin(y*pi)/10;
func vv = cos(x*pi)/10;

mesh Th = buildmesh ( a(6) + b(4) + c(4) + d(4) + e(4) + f(6));
plot(Th, wait=1, fill=1, ps="Lshape.eps"); // see figure 3.10
real coef=1;
real minT0= checkmovemesh(Th, [x, y]); // the min triangle area
while(1) // find a correct move mesh
{
    real minT=checkmovemesh(Th, [x+coef*uu, y+coef*vv]); // the min triangle area
    if (minT > minT0/5) break; // if big enough
    coef=/1.5;
}
```
Note 9  Consider a function \( u \) defined on a mesh \( \text{Th} \). A statement like \( \text{Th}=\text{movemesh}(\text{Th},...) \) does not change \( u \) and so the old mesh still exists. It will be destroyed when no function use it. A statement like \( u = u \) redefines \( u \) on the new mesh \( \text{Th} \) with interpolation and therefore destroys the old \( \text{Th} \) if \( u \) was the only function using it.

Example 13 (movemesh.edp)  Now, we given an example of moving mesh with lagrangian function \( u \) defined on the moving mesh.

```plaintext
// simple movemesh example
mesh Th=square(10,10);
fespace Vh(Th,P1);
real t=0;

// ---
// the problem is how to build data without interpolation
// so the data u is moving with the mesh as you can see in the plot
// ---
Vh u=y;
for (int i=0;i<4;i++)
{
  t=i*0.1;
  Vh f= x*t;
  real minarea=checkmovemesh(Th,[x,y+f]);
  if (minarea >0 )
    Th=movemesh(Th,[x,y+f]);
    cout << " Min area " << minarea << endl;
  real[int] tmp(u[].n);
  tmp=u[];
  // save the value
```
3.4 Regular Triangulation

For a set $S$, we define the diameter of $S$ by

$$diam(S) = \sup\{|x - y|; \ x, y \in S\}$$

The sequence $\{T_h\}_{h \downarrow 0}$ of $\Omega$ is called regular if they satisfy the following:

1. $$\lim_{h \downarrow 0} \max\{diam(T_k) | T_k \in T_h\}$$

2. There is a number $\sigma > 0$ independent of $h$ such that

$$\frac{\rho(T_k)}{diam(T_k)} \geq \sigma \text{ for all } T_k \in T_h$$

where $\rho(T_k)$ are the diameter of the inscribed circle of $T_k$.

We put $h(T_h) = \max\{diam(T_k) | T_k \in T_h\}$, which is obtained by

```cpp
mesh Th = .......;
fespace Ph(Th,P0);
Ph h = hTriangle;
cout << "size of mesh = " << h[].max << endl;
```

3.5 Adaptmesh

The function

$$f(x, y) = 10.0x^3 + y^3 + \tan^{-1}[\epsilon/(\sin(5.0y) - 2.0x)]$$

$\epsilon = 0.0001$

sharply vary its value. However, the initial mesh given by the command in Section 3.1 cannot reflect its sharp variation.

Example 14

```cpp
real eps = 0.0001;
real h=1;
real hmin=0.05;
func f = 10.0*x^3+y^3+h*atan2(eps,sin(5.0*y)-2.0*x);
```
3.5. ADAPTMESH

```plaintext
mesh Th = square(5,5,[−1+2*x,−1+2*y]);

fespace Vh(Th,P1);
Vh fh=f;
plot(fh);
for (int i=0;i<2;i++)
{
    Th = adaptmesh(Th,fh);
    fh=f;
    plot(Th,fh,wait=1);
}
```

Figure 3.12: 3D graph under the initial mesh and after of 1st and 2nd adaptation

freefem++ uses a variable metric/Delaunay automatic meshing algorithm. The command

```plaintext
mesh ATh = adaptmesh(Th,f);
```
create the new mesh ATh by the Hessian

\[
D^2f = \left( \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial y^2} \right)
\]

of a function (formula or FE-function). Mesh adaptation is a very powerful tool when the solution of a problem vary locally and sharply.

Here we solve the problem (1.1)-(1.2), when \( f = 1 \) and \( \Omega \) is L-shape domain.

Example 15 (Adapt.edp) The solution has the singularity \( r^{3/2} \), \( r = |x - \gamma| \) at the point \( \gamma \) of the intersection of two lines \( bc \) and \( bd \) (see Fig. 3.13).

```plaintext
border ba(t=0,1.0){x=t; y=0; label=1;};
border bb(t=0,0.5){x=1; y=t; label=1;};
border bc(t=0,0.5){x=1-t; y=0.5; label=1;};
border bd(t=0.5,1){x=0.5; y=t; label=1;};
border be(t=0.5,1){x=1-t; y=1; label=1;};
border bf(t=0.0,1){x=0; y=1-t; label=1;};
mesh Th = buildmesh ( ba(6)+bb(4)+bc(4)+bd(4)+be(4)+bf(6) );
fespace Vh(Th,P1);
Vh u,v;
```

// set FE space
// set unknown and test function
func f = 1;
real error=0.1;    // level of error
problem Poisson(u,v,solver=CG,eps=1.0e-6) =
    int2d(Th) ( dx(u)*dx(v) + dy(u)*dy(v))
- int2d(Th) ( f*v )
+ on(1,u=0) ;
for (int i=0;i< 4;i++)
{
    Poisson;
    Th=adaptmesh(Th,u,err=error);
    error = error/2;
}
plot(u);

To speed up the adaptation we change by hand a default parameter err of adaptmesh, which specifies the required precision, so as to make the new mesh finer. The problem is coercive and symmetric, so the linear system can be solved with the conjugate gradient method (parameter solver=CG with the stopping criteria on the residual, here eps=1.0e-6). By adaptmesh, we get good slope of the final solution near the point of intersection of bc and bd as in Fig. 3.14.

This method is described in detail in [7]. It has a number of default parameters which can be modified:

- **hmin** = Minimum edge size. (val is a real. Its default is related to the size of the domain to be meshed and the precision of the mesh generator).

- **hmax** = Maximum edge size. (val is a real. It defaults to the diameter of the domain to be meshed)

- **err** = $P^1$ interpolation error level (0.01 is the default).

- **errg** = Relative geometrical error. By default this error is 0.01, and in any case it must be lower than $1/\sqrt{2}$. Meshes created with this option may have some edges smaller than the −hmin due to geometrical constraints.
3.5. ADAPTMESH

nbvx= Maximum number of vertices generated by the mesh generator (9000 is the default).

nbsmooth= number of iterations of the smoothing procedure (5 is the default).

nbjacoby= number of iterations in a smoothing procedure during the metric construction, 0 means no smoothing (6 is the default).

ratio= ratio for a prescribed smoothing on the metric. If the value is 0 or less than 1.1 no smoothing is done on the metric (1.8 is the default).

If ratio > 1.1, the speed of mesh size variations is bounded by \( \log(\text{ratio}) \). Note: As ratio gets closer to 1, the number of generated vertices increases. This may be useful to control the thickness of refined regions near shocks or boundary layers.

omega= relaxation parameter for the smoothing procedure (1.0 is the default).

iso= If true, forces the metric to be isotropic (false is the default).

abseerror= If false, the metric is evaluated using the criterium of equi-repartition of relative error (false is the default). In this case the metric is defined by

\[
M = \left( \frac{1}{err\ coef^2} \frac{|H|}{\max(\text{CutOff}, |\eta|)} \right)^p \tag{3.1}
\]

otherwise, the metric is evaluated using the criterium of equi-distribution of errors. In this case the metric is defined by

\[
M = \left( \frac{1}{err\ coef^2} \frac{|H|}{\sup(\eta) - \inf(\eta)} \right)^p. \tag{3.2}
\]

cutoff= lower limit for the relative error evaluation (1.0e-6 is the default).

verbosity= informational messages level (can be chosen between 0 and \( \infty \)). Also changes the value of the global variable verbosity (obsolete).

inquire= To inquire graphically about the mesh (false is the default).

splitpbedge= If true, splits all internal edges in half with two boundary vertices (true is the default).

maxsubdiv= Changes the metric such that the maximum subdivision of a background edge is bound by \( \text{val} \) (always limited by 10, and 10 is also the default).

rescaling= if true, the function with respect to which the mesh is adapted is rescaled to be between 0 and 1 (true is the default).

keepbackvertices= if true, tries to keep as many vertices from the original mesh as possible (true is the default).
**isMetric**= if true, the metric is defined explicitly (false is the default). If the 3 functions $m_{11}, m_{12}, m_{22}$ are given, they directly define a symmetric matrix field whose Hessian is computed to define a metric. If only one function is given, then it represents the isotropic mesh size at every point.

For example, if the partial derivatives $f_{xx} (= \partial^2 f / \partial x^2)$, $f_{xy} (= \partial^2 f / \partial x \partial y)$, $f_{yy} (= \partial^2 f / \partial y^2)$ are given, we can set

$$\text{Th}= \text{adaptmesh}(\text{Th}, \text{fxx}, \text{fxy}, \text{fyy}, \text{IsMetric}=1, \text{nbvx}=10000, \text{hmin}=\text{hmin});$$

**power**= exponent power of the Hessian used to compute the metric (1 is the default).

**thetamax**= minimum corner angle in degrees (default is 0).

**splitin2**= boolean value. If true, splits all triangles of the final mesh into 4 sub-triangles.

**metric**= an array of 3 real arrays to set or get metric data information. The size of these three arrays must be the number of vertices. So if $m_{11}, m_{12}, m_{22}$ are three P1 finite elements related to the mesh to adapt, you can write: \text{metric}=[m_{11}[], m_{12}[], m_{22}[]] (see file convect-apt.edp for a full example)

**nomeshgeneration**= If true, no adapted mesh is generated (useful to compute only a metric).

**periodic**= As writing \text{periodic}=[[4,y],[2,y],[1,x],[3,x]]; it builds an adapted periodic mesh. The sample build a biperiodic mesh of a square. (see periodic finite element spaces 4, and see sphere.edp for a full example)

### 3.6 Trunc

A small operator to create a truncated mesh from a mesh with respect to a boolean function. The two named parameter

**label**= sets the label number of new boundary item (one by default)

**split**= sets the level $n$ of triangle splitting. each triangle is splitted in $n \times n$ (one by default).

To create the mesh Th3 where all triangles of a mesh Th are splitted in $3 \times 3$, just write:

$$\text{mesh Th3} = \text{trunc(Th,1,split=3);}$$

The truncmesh.edp example construct all ”trunc” mesh to the support of the basic function of the space $Vh$ (cf. abs(u)>0), split all the triangles in $5 \times 5$, and put a label number to 2 on new boundary.

```plaintext
mesh Th=square(3,3);
fespace Vh(Th,P1);
Vh u;
```
```c
int i,n=u.n;

for (i=0;i<n;i++) { // all degree of freedom
    u[i]=1; // the basic function i
    plot(u,wait=1); // plot the mesh of
    mesh Sh1=trunc(Th,abs(u)>1.e-10,split=5,label=2);
    plot(Th,Sh1,wait=1,ps="trunc+i+.eps"); // plot the mesh of
    u[i]=0; // reset
}
```

Figure 3.15: mesh of support the function $P_1$ number 0, splitted in $5 \times 5$

Figure 3.16: mesh of support the function $P_1$ number 6, splitted in $5 \times 5$

### 3.7 splitmesh

A other way to split mesh triangle:

```c
assert(version>=1.37); // new stuff 2004 splitmesh (version 1.37)

border a(t=0,2*pi){ x=cos(t); y=sin(t);label=1;}
plot(Th,wait=1,ps="nosplitmesh.eps"); // see figure 3.17
mesh Th=buildmesh(a(20));
plot(Th,wait=1);
Th=splitmesh(Th,1+5*(square(x-0.5)+y*y));
plot(Th,wait=1,ps="splitmesh.eps"); // see figure 3.18
```

3.8 A Fast Finite Element Interpolator

In practice one may discretize the variational equations by the Finite Element method. Then there will be one mesh for $\Omega_1$ and another one for $\Omega_2$. The computation of integrals of products of functions defined on different meshes is difficult. Quadrature formulae and interpolations from one mesh to another at quadrature points are needed. We present below the interpolation operator which we have used and which is new, to the best of our knowledge. Let $T^0_h = \cup_k T^0_k, T^1_h = \cup_k T^1_k$ be two triangulations of a domain $\Omega$. Let

$$V(T^i_h) = \{C^0(\Omega^i_h) : f|_{T^i_k} \in P^1\}, \quad i = 0, 1$$

be the spaces of continuous piecewise affine functions on each triangulation. Let $f \in V(T^0_h)$. The problem is to find $g \in V(T^1_h)$ such that

$$g(q) = f(q) \quad \forall q \text{ vertex of } T^1_h$$

Although this is a seemingly simple problem, it is difficult to find an efficient algorithm in practice. We propose an algorithm which is of complexity $N^1 \log N^0$, where $N^i$ is the number of vertices of $T^i_h$, and which is very fast for most practical 2D applications.

**Algorithm**

The method has 5 steps. First a quadtree is built containing all the vertices of mesh $T^0_h$ such that in each terminal cell there are at least one, and at most 4, vertices of $T^0_h$. For each $q^1$, vertex of $T^1_h$ do:

**Step 1** Find the terminal cell of the quadtree containing $q^1$. 

Figure 3.17: initial mesh  
Figure 3.18: all left mesh triangle is split conformally in int $(1+5*(\text{square}(x-0.5)+y*y))^2$ triangles.
Step 2 Find the nearest vertex \( q_j^0 \) to \( q^1 \) in that cell.

Step 3 Choose one triangle \( T_k^0 \in \mathcal{T}_h^0 \) which has \( q_j^0 \) for vertex.

Step 4 Compute the barycentric coordinates \( \{\lambda_j\}_{j=1,2,3} \) of \( q^1 \) in \( T_k^0 \).

- if all barycentric coordinates are positive, go to Step 5
- else if one barycentric coordinate \( \lambda_i \) is negative replace \( T_k^0 \) by the adjacent triangle opposite \( q_j^0 \) and go to Step 4.
- else two barycentric coordinates are negative so take one of the two randomly and replace \( T_k^0 \) by the adjacent triangle as above.

Step 5 Calculate \( g(q^1) \) on \( T_k^0 \) by linear interpolation of \( f \):

\[
g(q^1) = \sum_{j=1,2,3} \lambda_j f(q_j^0)
\]

End

Figure 3.19: To interpolate a function at \( q^0 \) the knowledge of the triangle which contains \( q^0 \) is needed. The algorithm may start at \( q^1 \in \mathcal{T}_h^0 \) and stall on the boundary (thick line) because the line \( q^0q^1 \) is not inside \( \Omega \). But if the holes are triangulated too (doted line) then the problem does not arise.

Two problems needs to solved:

- **What if \( q^1 \) is not in \( \Omega_h^0 \)?** Then Step 5 will stop with a boundary triangle. So we add a step which test the distance of \( q^1 \) with the two adjacent boundary edges and select the nearest, and so on till the distance grows.

- **What if \( \Omega_h^0 \) is not convex and the marching process of Step 4 locks on a boundary?** By construction Delaunay-Voronoi mesh generators always triangulate the convex hull of the vertices of the domain. So we make sure that this information is not lost when \( \mathcal{T}_h^0, \mathcal{T}_h^1 \) are constructed and we keep the triangles which are outside the domain in a special list. Hence in step 5 we can use that list to step over holes if needed.

**Note 10** Step 3 requires an array of pointers such that each vertex points to one triangle of the triangulation.
3.9 Meshing Examples

Example 16 (Two rectangles touching by a side)

```plaintext
border a(t=0,1){x=t; y=0;};
border b(t=0,1){x=1; y=t;};
border c(t=1,0){x=t; y=1;};
border d(t=1,0){x=0; y=t;};
border c1(t=0,1){x=t; y=1;};
border e(t=0,0.2){x=1; y=1+t;};
border f(t=1,0){x=t; y=1.2;};
border g(t=0.2,0){x=0; y=1+t;};

int n=1;

mesh th = buildmesh(a(10*n)+b(10*n)+c(10*n)+d(10*n));
mesh TH = buildmesh( c1(10*n) + e(5*n) + f(10*n) + g(5*n) );

plot(th,TH,ps="TouchSide.esp");
```

Figure 3.20: Two rectangles touching by a side

Example 17 (NACA0012 Airfoil)

```plaintext
border upper(t=0,1){ x = t; y = 0.17735*sqrt(t)-0.075597*t-0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); }
border lower(t=1,0){ x = t; y = -(0.17735*sqrt(t)-0.075597*t-0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4)); }
border c(t=0,2*pi){ x=0.8*cos(t)+0.5; y=0.8*sin(t); }

mesh Th = buildmesh(c(30)+upper(35)+lower(35));

plot(Th,ps="NACA0012.eps",bw=1);
```

Figure 3.21: NACA0012 Airfoil

Example 18 (Cardioid)

```plaintext
real b = 1, a = b;
border C(t=0,2*pi){ x=(a+b)*cos(t)-b*cos((a+b)*t/b);
```

Figure 3.20: Two rectangles touching by a side
3.9. MESHING EXAMPLES

\[ y = (a+b) \cdot \sin(t) - b \cdot \sin((a+b) \cdot t/b); \]

\[
\text{mesh } \text{Th} = \text{buildmesh}(C(50)); \\
\text{plot(Th, ps=}}"\text{Cardioid.eps"}, \text{bw=1}); \] // Fig. 3.22

Example 19 (Cassini Egg)

\[
\text{border } C(t=0,2 \pi) \{ \ x = (2 \cdot \cos(2 \cdot t)+3) \cdot \cos(t); \\
\quad \ y = (2 \cdot \cos(2 \cdot t)+3) \cdot \sin(t); \}
\]

\[
\text{mesh } \text{Th} = \text{buildmesh}(C(50)); \\
\text{plot(Th, ps=}}"\text{Cassini.eps"}, \text{bw=1}); \] // Fig. 3.23

Figure 3.22: Domain with Cardioid curve boundary

Figure 3.23: Domain with Cassini Egg curve boundary

Example 20 (By cubic Bezier curve)

\[
\text{// A cubic Bezier curve connecting two points with two control points}
\]

\[
\text{func real bzi(real p0, real p1, real q1, real q2, real t)} \\
\{ \quad \text{return } p0*(1-t)^3+q1*3*(1-t)^2*t+q2*3*(1-t)*t^2+p1*t^3; \}
\]

real[int] p00=[0,1], p01=[0,-1], q00=[-2,0.1], q01=[-2,-0.5];
real[int] p11=[1,-0.9], q10=[0.1,-0.95], q11=[0.5,-1];
real[int] p21=[2,0.7], q20=[3,-0.4], q21=[4,0.5];
real[int] q30=[0.5,1.1], q31=[1.5,1.2];
\[
\text{border G1(t=0,1) } \{ \ x = \text{bzi(p00[0],p01[0],q00[0],q01[0],t);} \\
\quad \ y = \text{bzi(p00[1],p01[1],q00[1],q01[1],t);} \} \\
\text{border G2(t=0,1) } \{ \ x = \text{bzi(p01[0],p11[0],q10[0],q11[0],t);} \\
\quad \ y = \text{bzi(p01[1],p11[1],q10[1],q11[1],t);} \} \\
\text{border G3(t=0,1) } \{ \ x = \text{bzi(p11[0],p21[0],q20[0],q21[0],t);} \\
\quad \ y = \text{bzi(p11[1],p21[1],q20[1],q21[1],t);} \} \\
\text{border G4(t=0,1) } \{ \ x = \text{bzi(p21[0],p00[0],q30[0],q31[0],t);} \\
\quad \ y = \text{bzi(p21[1],p00[1],q30[1],q31[1],t);} \}
\]

int m=5;
\[
\text{mesh Th} = \text{buildmesh(G1(2*m)+G2(m)+G3(3*m)+G4(m));} \\
\text{plot(Th, ps=}}"\text{Bezier.eps"}, \text{bw=1}); \] // Fig 3.24
Example 21 (Section of Engine)

\[
\text{real } a = 6., b = 1., c = 0.5; \\
\text{border } L1(t=0,1) \{ x = -a; y = 1 + b - 2 \cdot (1 + b) \cdot t; \} \\
\text{border } L2(t=0,1) \{ x = -a + 2 \cdot a \cdot t; y = -1 - b \cdot (x/a) \cdot (x/a) \cdot (3 - 2 \cdot \text{abs}(x)/a); \} \\
\text{border } L3(t=0,1) \{ x = a; y = -1 - b + (1 + b) \cdot t; \} \\
\text{border } L4(t=0,1) \{ x = a - a \cdot t; y = 0; \} \\
\text{border } L5(t=0,\pi) \{ x = -c \cdot \sin(t)/2; y = c/2 - c \cdot \cos(t)/2; \} \\
\text{border } L6(t=0,1) \{ x = a \cdot t; y = c; \} \\
\text{border } L7(t=0,1) \{ x = a; y = c + (1 + b - c) \cdot t; \} \\
\text{border } L8(t=0,1) \{ x = a - 2 \cdot a \cdot t; y = 1 + b \cdot (x/a) \cdot (x/a) \cdot (3 - 2 \cdot \text{abs}(x)/a); \} \\
\text{mesh } Th = \text{buildmesh}(L1(8) + L2(26) + L3(8) + L4(20) + L5(8) + L6(30) + L7(8) + L8(30)); \\
\text{plot}(Th, ps=\text{\textasciitilde Engine.eps}, bw=1); \text{ // Fig. 3.25}
\]

![Figure 3.24: Boundary drawn by Bezier curves](image)

![Figure 3.25: Section of Engine](image)

Example 22 (Domain with U-shape channel)

\[
\text{real } d = 0.1; \text{ // width of U-shape} \\
\text{border } L1(t=0,1-d) \{ x = -1; y = -d - t; \} \\
\text{border } L2(t=0,1-d) \{ x = -1; y = 1 - t; \} \\
\text{border } B(t=0,2) \{ x = -1 + t; y = -1; \} \\
\text{border } C1(t=0,1) \{ x = t - 1; y = d; \} \\
\text{border } C2(t=0,2 \cdot d) \{ x = 0; y = d - t; \} \\
\text{border } C3(t=0,1) \{ x = -t; y = -d; \} \\
\text{border } R(t=0,2) \{ x = 1; y = 1 - t; \} \\
\text{border } T(t=0,2) \{ x = 1 - t; y = 1; \} \\
\text{int } n = 5; \\
\text{mesh } Th = \text{buildmesh}((L1(n/2) + L2(n/2) + B(n) + C1(n) + C2(3) + C3(n) + R(n) + T(n))); \\
\text{plot}(Th, ps=\text{\textasciitilde U-shape.eps}, bw=1); \text{ // Fig 3.26}
\]

Example 23 (Domain with V-shape cut)

\[
\text{real } dAg = 0.01; \text{ // angle of V-shape} \\
\text{border } C(t=dAg,2 \cdot \pi - dAg) \{ x = \cos(t); y = \sin(t); \} \\
\text{real}[\text{int}] pa(2), pb(2), pc(2);
3.9. MESHING EXAMPLES

\[
p_{a_0} = \cos(d \cdot A_g); \quad p_{a_1} = \sin(d \cdot A_g);
\]
\[
p_{b_0} = \cos(2 \pi - d \cdot A_g); \quad p_{b_1} = \sin(2 \pi - d \cdot A_g);
\]
\[
p_{c_0} = 0; \quad p_{c_1} = 0;
\]
\[
\text{border} \quad \text{seg1}(t=0,1) \{ x=(1-t) \cdot p_{b_0} + t \cdot p_{c_0}; \quad y=(1-t) \cdot p_{b_1} + t \cdot p_{c_1}; \};
\]
\[
\text{border} \quad \text{seg2}(t=0,1) \{ x=(1-t) \cdot p_{c_0} + t \cdot p_{a_0}; \quad y=(1-t) \cdot p_{c_1} + t \cdot p_{a_1}; \};
\]
\[
\text{mesh} \quad \text{Th} = \text{buildmesh}(\text{seg1}(20)+\text{C}(40)+\text{seg2}(20));
\]
\[
\text{plot} \quad \text{Th}, \text{ps}="V-shape.eps", \text{bw}=1); \quad // \quad \text{Fig. 3.27}
\]

Figure 3.26: Domain with U-shape channel

changed by \( \alpha \)

Figure 3.27: Domain with V-shape cut

changed by \( d \cdot A_g \)

Example 24 (Smiling face)

\[
\text{real} \quad d=0.1;
\]
\[
\text{int} \quad m=5;
\]
\[
\text{real} \quad a=1.5, b=2, c=0.7, e=0.01;
\]
\[
\text{border} \quad \text{F}(t=0,2 \pi) \{ x=a \cdot \cos(t); \quad y=b \cdot \sin(t); \}
\]
\[
\text{border} \quad \text{E1}(t=0,2 \pi) \{ x=0.2 \cdot \cos(t)-0.5; \quad y=0.2 \cdot \sin(t)+0.5; \}
\]
\[
\text{border} \quad \text{E2}(t=0,2 \pi) \{ x=0.2 \cdot \cos(t)+0.5; \quad y=0.2 \cdot \sin(t)+0.5; \}
\]
\[
\text{func} \quad \text{real} \quad \text{st}(\text{real} \quad \text{t}) \{ \text{return} \quad \sin(\text{pi} \cdot \text{t})-\pi/2; \}
\]
\[
\text{border} \quad \text{C1}(t=-0.5,0.5) \{ x=(1-d) \cdot c \cdot \cos(\text{st}(\text{t})); \quad y=(1-d) \cdot c \cdot \sin(\text{st}(\text{t})); \}
\]
\[
\text{border} \quad \text{C2}(t=0,1) \{ x=((1-d)+d \cdot t) \cdot c \cdot \cos(\text{st}(0.5));y=((1-d)+d \cdot t) \cdot c \cdot \sin(\text{st}(0.5)); \}
\]
\[
\text{border} \quad \text{C3}(t=0.5,-0.5) \{ x=c \cdot \cos(\text{st}(\text{t})); \quad y=c \cdot \sin(\text{st}(\text{t})); \}
\]
\[
\text{border} \quad \text{C4}(t=0,1) \{ x=(1-d) \cdot c \cdot \cos(\text{st}(-0.5)); \quad y=(1-d) \cdot c \cdot \sin(\text{st}(-0.5)); \}
\]
\[
\text{border} \quad \text{C0}(t=0,2 \pi) \{ x=0.1 \cdot \cos(t); \quad y=0.1 \cdot \sin(t); \}
\]
\[
\text{mesh} \quad \text{Th=buildmesh}(\text{F}(10 \cdot m)+\text{C1}(2 \cdot m)+\text{C2}(3)+\text{C3}(2 \cdot m)+\text{C4}(3)\nonumber
\]
\[+\text{C0}(m)+\text{E1}(-2 \cdot m)+\text{E2}(-2 \cdot m));\]
\[
\text{plot} \quad \text{Th,ps="SmileFace.eps",bw=1}); \quad // \quad \text{see Fig. 3.28}
\]
Example 25 (3point bending)

// Square for Three-Point Bend Specimens fixed on Fix1, Fix2
// It will be loaded on Load.

real a=1, b=5, c=0.1;
int n=5, m=b*n;
border Left(t=0,2*a) { x=-b; y=a-t; }
border Bot1(t=0,b/2-c) { x=-b+t; y=-a; }
border Fix1(t=0,2*c) { x=-b/2-c+t; y=-a; }
border Bot2(t=0,b-2*c) { x=-b/2+c+t; y=-a; }
border Fix2(t=0,2*c) { x=b/2-c+t; y=-a; }
border Bot3(t=0,b/2-c) { x=b/2+c+t; y=-a; }
border Right(t=0,2*a) { x=b; y=-a+t; }
border Top1(t=0,b-c) { x=b-t; y=a; }
border Load(t=0,2*c) { x=c-t; y=a; }
border Top2(t=0,b-c) { x=-c-t; y=a; }
mesh Th = buildmesh(Left(n)+Bot1(m/4)+Fix1(5)+Bot2(m/2)+Fix2(5)+Bot3(m/4)
                +Right(n)+Top1(m/2)+Load(10)+Top2(m/2));
plot(Th,ps="ThreePoint.eps",bw=1);

Figure 3.29: Domain for three-point bending test

Figure 3.28: Smiling face (Mouth is changeable)
Chapter 4

Finite Elements

As stated in Step 2 in Section 1.2, FEM makes approximations all functions \( w \) to

\[
w(x, y) \simeq w_0 \phi_0(x, y) + w_1 \phi_1(x, y) + \cdots + w_{M-1} \phi_{M-1}(x, y)
\]

with finite basis functions \( \phi_k(x, y) \) and numbers \( w_k \) \((k = 0, \cdots, M - 1)\). The functions \( \phi_k(x, y) \) is constructed from the triangle \( T_k \), so \( \phi_k(x, y) \) is called shape function. The finite element space

\[
V_h = \{ w | w_0 \phi_0 + w_1 \phi_1 + \cdots + w_{M-1} \phi_{M-1}, w_i \in \mathbb{R} \}
\]

is easily created by

```c
fespace IDspace(IDmesh,<IDFE>) ;
```

or with \( \ell \) pair of periodic boundary condition

```c
fespace IDspace(IDmesh,<IDFE>, periodic=[[la_1,sa_1],[lb_1,sb_1]],

... [la_k,sa_k],[lb_k,sb_k]]) ;
```

where \( \text{IDspace} \) is the name of the space (e.g. \( V_h \)), \( \text{IDmesh} \) is the name of the associated mesh and \( <\text{IDFE}> \) is a identifier of finite element type. In a pair of periodic boundary condition, if \( [1a_i,sa_i],[1b_i,sb_i] \) is a pair of int, this expresses the 2 labels \( 1a_i \) and \( 1b_i \) of the piece of the boundary to be equivalence; If \( [1a_i,sa_i],[1b_i,sb_i] \) is a pair of real, this expresses \( sa_i \) and \( sb_i \) give two common abscissa on the two boundary curve, and two points are identify if the two abscissa are equal.

As of today, the known types of finite element are:

**P0** piecewise constant discontinuous finite element

\[
P_{0h} = \{ v \in L^2(\Omega) \mid \text{for all } K \in T_h \text{ there is } \alpha_K \in \mathbb{R} : v|_K = \alpha_K \} \quad (4.1)
\]

**P1** piecewise linear continuous finite element

\[
P_{1h} = \{ v \in H^1(\Omega) \mid \forall K \in T_h \quad v|_K \in P_1 \} \quad (4.2)
\]

**P1dc** piecewise linear discontinuous finite element

\[
P_{1dc} = \{ v \in L^2(\Omega) \mid \forall K \in T_h \quad v|_K \in P_1 \} \quad (4.3)
\]
CHAPTER 4. FINITE ELEMENTS

P1b piecewise linear continuous finite element plus bubble

\[ P1b_h = \{ v \in H^1(\Omega) \mid \forall K \in T_h \quad v|_K \in P_1 \oplus \text{Span}\{\lambda^K_0, \lambda^K_1, \lambda^K_2\} \} \]  

(4.4)

where \( \lambda^K_i, i = 0, 1, 2 \) are the 3 area coordinate functions of the triangle \( K \).

P2 piecewise \( P_2 \) continuous finite element,

\[ P2_h = \{ v \in H^1(\Omega) \mid \forall K \in T_h \quad v|_K \in P_2 \} \]  

(4.5)

where \( P_2 \) is the set of polynomials of \( \mathbb{R}^2 \) of degrees \( \leq 2 \).

P2dc piecewise \( P_2 \) discontinuous finite element,

\[ P2dc_h = \{ v \in L^2(\Omega) \mid \forall K \in T_h \quad v|_K \in P_2 \} \]  

(4.6)

RT0 Raviart-Thomas finite element

\[ RT0_h = \{ v \in H(\text{div}) \mid \forall K \in T_h \quad v|_K(x,y) = |^{\alpha^K}_{\beta^K} + |^{\gamma^K}_y \} \]  

(4.7)

where by writing \( \text{div} \; w = \partial w_1/\partial x + \partial w_2/\partial y, \; w = (w_1, w_2), \)

\( H(\text{div}) = \{ w \in L^2(\Omega)^2 \mid \text{div} \; w \in L^2(\Omega) \} \)

and \( \alpha^K, \beta^K, \gamma^K \) are real numbers.

P1nc piecewise linear element continuous at the middle of edge only.

If we get the finite element spaces

\[ X_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \quad v|_K \in P_1 \} \]

\[ X_{ph} = \{ v \in X_h \mid v|_0 = v|_1 \}
\]

\[ M_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \quad v|_K \in P_2 \} \]

\[ R_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \quad v|_K(x,y) = |^{\alpha^K}_{\beta^K} + |^{\gamma^K}_y \} \]

when \( T_h \) is a mesh \( 10 \times 10 \) of the unit square \([0,1]^2\), we only write in \texttt{freefem++} as follows:

```plaintext
mesh Th=square(10,10);
fespace Xh(Th,P1); // scalar FE
fespace Xph(Th,P1,
pervid=[2,y],[4,y],[1,x],[3,x])); // bi-periodic FE
fespace Mh(Th,P2); // scalar FE
fespace Rh(Th,RT0); // vectorial FE
```

where \( \text{Xh, Mh, Rh} \) expresses finite element spaces (called FE spaces) \( X_h, M_h, R_h \), respectively. If we want use FE-functions \( u_h, v_h \in X_h \) and \( p_h, q_h \in M_h \) and \( U_h, V_h \in R_h \), we write in \texttt{freefem++}

\[ Xh \; \text{uh, vh; } \]
\[ Xph \; \text{uph, vph; } \]
\[ Mh \; \text{ph, qh; } \]
\[ Rh \; \text{[Uxh, Uyh], [Vxh, Vyh]; } \]
\[ Xh[\text{int}] \; \text{Uh(10); } \]
\[ Rh[\text{int}] \; \text{[Wxh, Wyh](10); } \]

The functions \( U_h, V_h \) have two components so we have

\[ U_h = |^{U_{xh}}_{U_{yh}} \quad \text{and} \quad V_h = |^{V_{xh}}_{V_{yh}} \]
4.1 Lagrange finite element

4.1.1 P0-element
For each triangle \( T_k \), the basis function \( \phi_k \) in \( Vh(Th, P0) \) is given by
\[
\phi_k(x, y) = 1 \text{ if } (x, y) \in T_k, \quad \phi_k(x, y) = 0 \text{ if } (x, y) \notin T_k
\]
If we write
\[
Vh(Th, P0); \quad \forall h f(h) = f(x, y);
\]
then for vertices \( q^k_i, i = 1, 2, 3 \) in Fig. 4.1(a),
\[
f_h = f_h(x, y) = \frac{1}{3} \sum_{k=1}^{n_v} f(q^k_1) + f(q^k_2) + f(q^k_3) \phi_k
\]
See Fig. 4.3 for the projection of \( f(x, y) = \sin(\pi x) \cos(\pi y) \) into \( Vh(Th, P0) \) when the mesh \( Th \) with \( 4 \times 4 \)-grid of \([-1, 1]^2 \) as in Fig. 4.2.

4.1.2 P1-element

For each vertex \( q^i \), the basis function \( \phi_i \) in \( Vh(Th, P1) \) is given by
\[
\phi_i(x, y) = a_i^k + b_i^k x + c_i^k y \text{ for } (x, y) \in T_k, \quad \phi_i(q^i) = 1, \quad \phi_i(q^j) = 0 \text{ if } i \neq j
\]
The basis function \( \phi_{k1}(x, y) \) with the vertex \( q^{k1} \) in Fig. 4.1(a) at point \( p = (x, y) \) in triangle \( T_k \) simply coincide with the barycentric coordinates \( \lambda^k_1 \) (area coordinates) :
\[
\phi_{k1}(x, y) = \lambda^k_1(x, y) = \frac{\text{area of triangle}(p, q^{k2}, q^{k3})}{\text{area of triangle}(q^{k1}, q^{k2}, q^{k3})}
\]
If we write
\[
Vh(Th, P1); \quad \forall h f(h) = g(x, y);
\]
then
\[
f_h = f_h(x, y) = \sum_{i=1}^{n_v} f(q^i) \phi_i(x, y)
\]
See Fig. 4.4 for the projection of \( f(x, y) = \sin(\pi x) \cos(\pi y) \) into \( Vh(Th, P1) \).
4.1.3 P2-element

For each vertex or midpoint $q^i$, the basis function $\phi_i$ in $V_h(Th, P_2)$ is given by
\[
\phi_i(x, y) = a_k^i + b_k^i x + c_k^i y + d_k^i x^2 + e_k^i x y + f_k^i y^2 \text{ for } (x, y) \in T_k,
\]
\[
\phi_i(q^i) = 1, \quad \phi_i(q^j) = 0 \text{ if } i \neq j
\]
The basis function $\phi_{k_1}(x, y)$ with the vertex $q_{k_1}$ in Fig. 4.1(b) is defined by the barycentric coordinates:
\[
\phi_{k_1}(x, y) = \lambda_{k_1}^k(x, y)(2\lambda_{k_1}^k(x, y) - 1)
\]
and for the midpoint $q_{k_2}$
\[
\phi_{k_2}(x, y) = 4\lambda_{k_1}^k(x, y)\lambda_{k_4}^k(x, y)
\]
If we write
\[
V_h(Th, P_2); \quad f_h = f(x, y);
\]
then
\[
f_h = f_h(x, y) = \sum_{i=1}^{M} f(q^i)\phi_i(x, y) \quad (\text{summation over all vertex or midpoint})
\]
See Fig. 4.5 for the projection of $f(x, y) = \sin(\pi x) \cos(\pi y)$ into $V_h(Th, P_2)$.

4.2 P1 Nonconforming Element

Refer [13] for detail. In Section 4.1, the approximation are a continuous function all over the domain, and
\[
w_h \in V_h \subset H^1(\Omega)
\]
However, we allow the continuity requirement to be relaxed. If we write
\[
V_h(Th, P_{linc}); \quad f_h = f(x, y);
\]
then
\[
f_h = f_h(x, y) = \sum_{i=1}^{n_m} f(m^i)\phi_i(x, y) \quad (\text{summation over all midpoint})
\]
4.2. P1 NONCONFORMING ELEMENT

Here the basis function $\phi_i$ associated with the midpoint $m^i = (q^k_i + q^{k+1}_i)/2$ where $q^k_i$ is the $i$-th point in $T_k$, and we assume that $j + 1 = 0$ if $j = 3$:

$$
\phi_i(x, y) = a_k^i + b_k^i x + c_k^i y \text{ for } (x, y) \in T_k,
\phi_i(m^i) = 1, \quad \phi_i(m^j) = 0 \text{ if } i \neq j
$$

Strictly speaking $\partial \phi_i / \partial x, \partial \phi_i / \partial y$ contain Dirac distribution $\rho \delta_{\partial T_k}$. The numerical calculations will automatically ignore them. In [18], there is a proof of the estimation

$$
\left( \sum_{k=1}^{n_v} \int_{T_k} |\nabla w - \nabla w_h|^2 dxdy \right)^{1/2} = O(h)
$$

The basis functions $\phi_k$ have the following properties.

1. For the bilinear form $a$ defined in (1.5) satisfy

$$
a(\phi_i, \phi_i) > 0, \quad a(\phi_i, \phi_j) \leq 0 \quad \text{if } i \neq j
$$

$$
\sum_{k=1}^{n_v} a(\phi_i, \phi_k) \geq 0
$$

2. $f \geq 0 \Rightarrow u_h \geq 0$

3. If $i \neq j$, the basis function $\phi_i$ and $\phi_j$ are $L^2$-orthogonal:

$$
\int_{\Omega} \phi_i \phi_j dxdy = 0 \quad \text{if } i \neq j
$$

which is false for $P_1$-element.

See Fig. 4.6 for the projection of $f(x, y) = \sin(\pi x) \cos(\pi y)$ into $v_h(Th, P_{1nc})$. See Fig. 4.6 for the projection of $f(x, y) = \sin(\pi x) \cos(\pi y)$ into $v_h(Th, P_{1nc})$. 

Figure 4.4: projection to $V_h(Th, P_1)$

Figure 4.5: projection to $V_h(Th, P_2)$
4.3 Other FE-space

For each triangle $T_k \in \mathcal{T}_h$, let $\lambda_{k1}(x,y)$, $\lambda_{k2}(x,y)$, $\lambda_{k3}(x,y)$ be the area coordinate of the triangle (see Fig. 4.1), and put

$$\beta_k(x,y) = 27\lambda_{k1}(x,y)\lambda_{k2}(x,y)\lambda_{k3}(x,y) \quad (4.8)$$

called bubble function on $T_k$. The bubble function has the feature:

1. $\beta_k(x,y) = 0$ if $(x,y) \in \partial T_k$.
2. $\beta_k(q^{kb}) = 1$ where $q^{kb}$ is the barycentre $\frac{q^{k1}+q^{k2}+q^{k3}}{3}$.

If we write

$$V_h(Th, P_1b) \ ; \ V_h f_h=f(x,y);$$

then

$$f_h = f_h(x,y) = \sum_{i=1}^{n_v} f(q^i)\phi_i(x,y) + \sum_{k=1}^{n_t} f(q^{kb})\beta_k(x,y)$$

See Fig. 4.7 for the projection of $f(x,y) = \sin(\pi x)\cos(\pi y)$ into $V_h(Th, P_1b)$.

4.4 Vector valued FE-function

Functions from $\mathbb{R}^2$ to $\mathbb{R}^N$ with $N = 1$ is called scalar function and called vector valued when $N > 1$. When $N = 2$

```latex
fespace Vh(Th, [P0,P1]) ;
```

make the space

$$V_h = \{ w = (w_1, w_2) \mid w_1 \in V_h(T_h, P_0), \ w_2 \in V_h(T_h, P_1) \}$$
4.4. VECTOR VALUED FE-FUNCTION

4.4.1 Raviant-Thomas element

In the Raviart-Thomas finite element $RT_0_h$, the degree of freedom are the flux throw an edge $e$ of the mesh, where the flux of the function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is $\int_e f \cdot n_e$, $n_e$ is the unit normal of edge $e$.

This implies an orientation of all the edges of the mesh, for example we can use the global numbering of the edge vertices and we just go to small to large number.

To compute the flux, we use an quadrature formulation with one point, the middle point of the edge. Consider a triangle $T_k$ with three vertices $(a, b, c)$. Let denote the vertices numbers by $i_a, i_b, i_c$, and define the three edge vectors $e_1, e_2, e_3$ by $sgn(i_b - i_c)(b - c)$, $sgn(i_c - i_a)(c - a)$, $sgn(i_a - i_b)(a - b)$.

We get three basis functions,

$$
\phi_1^k = \frac{sgn(i_b - i_c)}{2|T_k|} (x - a), \quad \phi_2^k = \frac{sgn(i_c - i_a)}{2|T_k|} (x - b), \quad \phi_3^k = \frac{sgn(i_a - i_b)}{2|T_k|} (x - c),
$$

(4.9)

where $|T_k|$ is the area of the triangle $T_k$. If we write

$$
Vh(Th, RT0); \quad Vh [f1h, f2h] = [f1(x,y), f2(x,y)];
$$

then

$$
f_h = f_h(x,y) = \sum_{k=1}^{n_3} \sum_{l=1}^{6} n_{i_l j_l} |e^h| f_{j_l}(m^{il}) \phi_{i_l j_l}
$$

where $n_{i_l j_l}$ is the $j_l$-th component of the normal vector $n_i$,

$$
\{m_1, m_2, m_3\} = \left\{ \frac{b + c}{2}, \frac{a + c}{2}, \frac{b + a}{2} \right\}
$$

and $i_l = \{1,1,2,2,3,3\}$, $j_l = \{1,2,1,2,1,2\}$ with the order of $l$.

![Figure 4.8: normal vectors of each edge](image_url)

```
Example 26 mesh Th=square(2,2);
fespace Xh(Th,P1);
fespace Vh(Th,RT0);
Xh uh,vh;
Vh [Uxh,Uyh];
[Uxh,Uyh] = [sin(x),cos(y)]; // ok vectorial FE function
vh= x^2+y^2; // vh
```
\texttt{Th = square(5,5);}  \hspace{1cm}  \text{// change the mesh}  \\
\texttt{uh = x^2+y^2;}  \hspace{1cm}  \text{// Xh is unchange}  \\
\texttt{Uxh = x;}  \hspace{1cm}  \text{// error: impossible to set only 1 component}  \\
\texttt{vh = Uxh;}  \hspace{1cm}  \text{// of a vector FE function.}  \\
\texttt{plot(uh,ps="onoldmesh.eps");}  \hspace{1cm}  \text{// compute on the new Xh}  \\
\texttt{uh = uh;}  \hspace{1cm}  \text{// and now uh use the 5x5 mesh}  \\
\texttt{plot(uh,ps="onnewmesh.eps");}  \hspace{1cm}  \text{// figure 4.9}  \\
\texttt{vh((x-1/2,y))= x^2 + y^2;}  \hspace{1cm}  \text{// figure 4.10}  \\
\texttt{vh = Uxh;}  \hspace{1cm}  \text{// error: impossible to set only 1 component}  \\
\texttt{uh = uh;}  \hspace{1cm}  \text{// of a vector FE function.}  \\
\texttt{plot(uh,ps="onoldmesh.eps");}  \hspace{1cm}  \text{// compute on the new Xh}  \\
\texttt{uh = uh;}  \hspace{1cm}  \text{// and now uh use the 5x5 mesh}  \\
\texttt{plot(uh,ps="onnewmesh.eps");}  \hspace{1cm}  \text{// figure 4.9}  \\
\texttt{vh((x-1/2,y))= x^2 + y^2;}  \hspace{1cm}  \text{// figure 4.10}  \\

To get the value at a point \( x = 1, y = 2 \) of the FE function \( uh \), or \( [Uxh, Uyh] \), one writes

\begin{verbatim}
real value;
value = uh(2,4);  \hspace{1cm}  \text{// get value= uh(2,4)}
value = Uxh(2,4); \hspace{1cm}  \text{// get value= Uxh(2,4)}
value = uh; \hspace{1cm}  \text{// get value= uh(1,2)}
value = Uxh; \hspace{1cm}  \text{// get value= Uxh(1,2)}
value = Uyh; \hspace{1cm}  \text{// get value= Uyh(1,2)}.
\end{verbatim}

To get the value of the array associated to the FE function \( uh \), one writes

\begin{verbatim}
real value = uh[0];  \hspace{1cm}  \text{// get the value of degree of freedom 0}
real maxdf = uh[].max;  \hspace{1cm}  \text{// maximum value of degree of freedom}
int size = uh.n;  \hspace{1cm}  \text{// the number of degree of freedom}
real[int] array(uh.n) = uh[];  \hspace{1cm}  \text{// copy the array of the function uh}
\end{verbatim}

Warning for no scalar finite element function \( [Uxh, Uyh] \) the two array \( Uxh[] \) and \( Uyh[] \) are the same array, because the degree of freedom can touch more than one component. The other way to set a FE function is to solve a ‘problem’ (see below).
4.5 Problem and solve

For freefem++ a problem must be given in variational form, so we need a bilinear form \( a(u,v) \), a linear form \( \ell(f,v) \), and possibly a boundary condition form must be added.

\[
\text{problem } P(u,v) = a(u,v) - \ell(f,v) + \text{(boundary condition)};
\]

For example, see (1.4).

**Note 11** When you want to formulate the problem and to solve it in the same time, you can use the keyword solve,

4.6 Parameter Description for solve and problem

The parameters are FE function real or complex, the number \( n \) of parameters is even \( (n = 2 \times k) \), the \( k \) first function parameters are unknown, and the \( k \) last are test functions.

**Note 12** If the functions are a part of vectoriel FE then you must give all the functions of the vectorial FE in the same order (see laplaceMixte problem for example).

**Note 13** Don’t mixte complex and real parameters FE function.

**Bug: 1** The mixing of fespace with different periodic boundary condition is not implemented. So all the finite element space use for test or unknow functions in a problem, must have the same type of periodic boundary condition or no periodic boundary condition. No clean message is given and the result is impredictible, Sorry.

The named parameters are:

- **solver**= LU, CG, Crout, Cholesky, GMRES, UMFPACK ...

  The default solver is LU. The storage mode of the matrix of the underlying linear system depends on the type of solver chosen: for LU the matrix is sky-line non symmetric, for Crout the matrix is sky-line symmetric, for Cholesky the matrix is sky-line symmetric positive definite, for CG the matrix is sparse symmetric positive, and for GMRES or UMFPACK the matrix is just sparse.

- **eps**= a real expression. \( \varepsilon \) sets the stopping test for the iterative methods like CG. Note that if \( \varepsilon \) is negative then the stopping test is:

  \[
  ||Ax - b|| < |\varepsilon|
  \]

  if it is positive then the stopping test is

  \[
  ||Ax - b|| < \frac{|\varepsilon|}{||Ax_0 - b||}
  \]
CHAPTER 4. FINITE ELEMENTS

\textbf{init}\textup{=} boolean expression, if it is false or 0 the matrix is reconstructed. Note that if the mesh changes the matrix is reconstructed too.

\textbf{precon}\textup{=} name of a function (for example \texttt{P}) to set the preconditioner. The prototype for the function \texttt{P} must be

\begin{verbatim}
func real[int] P(real[int] & xx) ;
\end{verbatim}

\textbf{tgv}\textup{=} Huge value (10^{30}) used to lock boundary conditions (see (1.8))

4.7 Problem definition

Below \(v\) is the unknown function and \(w\) is the test function.

After the “=” sign, one may find sums of:

- a name; this is the name given to the variational form (type \texttt{varf}) for possible reuse.

- the bilinear form term: for given functions \(K\) and unknown function \(v\), test functions \(w\),

\begin{align*}
\text{-) } \text{int2d}(\text{Th})( K \ast v \ast w) &= \sum_{T \in \text{Th}} \int_T K \, v \, w \\
\text{-) } \text{int2d}(\text{Th},1)( K \ast v \ast w) &= \sum_{T \in \text{Th}, T \subset \Omega_1} \int_T K \, v \, w \\
\text{-) } \text{int1d}(\text{Th},2,5)( K \ast v \ast w) &= \sum_{T \in \text{Th}} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K \, v \, w \\
\text{-) } \text{intalledges}(\text{Th})( K \ast v \ast w) &= \sum_{T \in \text{Th}} \int_{\partial T} K \, v \, w \\
\text{-) } \text{intalledges}(\text{Th},1)( K \ast v \ast w) &= \sum_{T \in \text{Th}, T \subset \Omega_1} \int_{\partial T} K \, v \, w
\end{align*}

- they become a sparse matrix of type \texttt{matrix}

- the linear form term: for given functions \(K\), \(f\) and test functions \(w\),

\begin{align*}
\text{-) } \text{int1d}(\text{Th})( K \ast w) &= \sum_{T \in \text{Th}} \int_T K \, w \\
\text{-) } \text{int1d}(\text{Th},2,5)( K \ast w) &= \sum_{T \in \text{Th}} \int_{(\partial T \cup \Gamma) \cap (\Gamma_2 \cup \Gamma_5)} K \, w \\
\text{-) } \text{intalledges}(\text{Th})( f \ast w) &= \sum_{T \in \text{Th}} \int_{\partial T} f \, w
\end{align*}

- a vector of type \texttt{real[int]}

- The boundary condition form term:
4.8. NUMERICAL INTEGRATION

- An "on" form (for Dirichlet): on(1, u = g)
- A linear form on \( \Gamma \) (for Neumann): \(- \int_{\Gamma(1)} (f \ast w) \) or \(- \int_{\Gamma(3)} (f \ast w) \)
- A bilinear form on \( \Gamma \) or \( \Gamma_2 \) (for Robin): \( \int_{\Gamma(1)} (K \ast v \ast w) \) or \( \int_{\Gamma(2)} (K \ast v \ast w) \).

If needed, the different kind of terms in the sum can appear more than once.

Remark: the integral mesh and the mesh associated to test function or unknown function can be different in the case of linear form.

**Note 14** \( N.x \) and \( N.y \) are the normal's components.

**Important**: it is not possible to write in the same integral the linear part and the bilinear part such as in \( \int_{\Gamma(1)} (K \ast v \ast w - f \ast w) \).

### 4.8 Numerical Integration

Let \( D \) be a \( N \)-dimensional bounded domain. For an arbitrary polynomials \( f \) of degree \( r \), if we can find particular points \( \xi_j, j = 1, \ldots, J \) in \( D \) and constants \( \omega_j \) such that

\[
\int_D f(x) = \sum_{\ell=1}^L c_\ell f(\xi_\ell) \quad (4.10)
\]

then we have the error estimation (see Crouzeix-Mignot (1984)), then there exists a constant \( C > 0 \) such that,

\[
\left| \int_D f(x) - \sum_{\ell=1}^L \omega_\ell f(\xi_\ell) \right| \leq C |D|h^{r+1} \quad (4.11)
\]

for any function \( r + 1 \) times continuously differentiable \( f \) in \( D \), where \( h \) is the diameter of \( D \) and \( |D| \) its measure.

A point in the segment \([q_iq_j]\) is given as

\[
\{(x,y) \mid x = (1-t)q_i + tq_j, y = (1-t)q'_i + tq'_j, 0 \leq t \leq 1\}
\]

For a domain \( \Omega_h = \sum_{k=1}^n T_k \), \( T_h = \{T_k\} \), we can calculate the integral over \( \Gamma_h = \partial \Omega_h \) by

\[
\int_{\Gamma_h} f(x)ds = \text{int1d(Th)}(f)
= \text{int1d(Th, qfe=*)}(f)
= \text{int1d(Th, qforder=*)}(f)
\]

where * stands for the name of quadrature formulas or the order of the Gauss formula. Where \(|q_iq_j|\) is the length of segment \( q_iq_j \). For a part \( \Gamma_1 \) of \( \Gamma_h \) with the label "1", we can calculate the integral over \( \Gamma_1 \) by

\[
\int_{\Gamma_1} f(x,y)ds = \text{int1d(Th, 1)}(f)
= \text{int1d(Th, 1, qfe=qf2pE)}(f)
\]
The integral over $\Gamma_1$, $\Gamma_3$ are given by

$$\int_{\Gamma_1 \cup \Gamma_3} f(x,y) ds = \text{int1d}(Th,1,3)(f)$$

For each triangle $T_k = [q^{k_1}q^{k_2}q^{k_3}]$, the point $P(x,y)$ in $T_k$ is expressed by the area coordinate as $P(\xi,\eta)$:

$$|T_k| = \begin{vmatrix} 1 & q_x^{k_1} & q_y^{k_1} \\ 1 & q_x^{k_2} & q_y^{k_2} \\ 1 & q_x^{k_3} & q_y^{k_3} \end{vmatrix} \quad D_1 = \begin{vmatrix} 1 & x & y \\ 1 & q_x^{k_2} & q_y^{k_2} \\ 1 & q_x^{k_3} & q_y^{k_3} \end{vmatrix} \quad D_2 = \begin{vmatrix} 1 & x & y \\ 1 & q_x^{k_1} & q_y^{k_1} \\ 1 & q_x^{k_3} & q_y^{k_3} \end{vmatrix} \quad D_3 = \begin{vmatrix} 1 & x & y \\ 1 & q_x^{k_1} & q_y^{k_1} \\ 1 & q_x^{k_2} & q_y^{k_2} \end{vmatrix}$$

$$\xi = D_1/|T_k| \quad \eta = D_2/|T_k| \quad \text{then } 1 - \xi - \eta = D_3/|T_k|$$

For a domain $\Omega_h = \sum_{k=1}^{n} T_k$, $Th = \{T_k\}$, we can calculate the integral over $\Omega_h$ by

$$\int_{\Omega_h} f(x,y) = \text{int2d}(Th)(f)$$

$$= \text{int2d}(Th, qft=*)(f)$$

$$= \text{int2d}(Th, qforder=*)(f)$$

where * stands for the name of quadrature formulas or the order of the Gauss formula.

### 4.9 Variational Form, Sparse Matrix, Right Hand Side Vector

It is possible to define variational forms:

```
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);

varf bx(u1,q) = int2d(Th) ( (dx(u1)*q));

bx(u1,q) = \int_{\Omega_h} \frac{\partial u_1}{\partial x} q
```

```
varf by(u1,q) = int2d(Th) ( (dy(u1)*q));

by(u1,q) = \int_{\Omega_h} \frac{\partial u_1}{\partial y} q
```
### 4.9. Variational Form, Sparse Matrix, Right Hand Side Vector

<table>
<thead>
<tr>
<th>L</th>
<th>qfe=</th>
<th>qforder=</th>
<th>point in $T_k$</th>
<th>$\omega_k$</th>
<th>degree of exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>qf1pT</td>
<td>2</td>
<td>$(\frac{1}{3}, \frac{1}{3})$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>3</td>
<td>qf2pT</td>
<td>3</td>
<td>$(\frac{7}{2}, \frac{7}{2})$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>7</td>
<td>qf5pT</td>
<td>6</td>
<td>$(\frac{3}{2}, \frac{3}{2})$</td>
<td>$0.225</td>
<td>T_k</td>
</tr>
<tr>
<td>3</td>
<td>qf1pT1ump</td>
<td></td>
<td>$(0, 0)$</td>
<td>$</td>
<td>T_k</td>
</tr>
<tr>
<td>9</td>
<td>qf2pT4P1</td>
<td></td>
<td>$(\frac{1}{3}, \frac{1}{3})$</td>
<td>$</td>
<td>T_k</td>
</tr>
</tbody>
</table>

**Note 15** the parameters of the variational form are completely formal, but not the case in problem and solve functionality.

\[
a(u_1, u_2) = \text{int2d}(Th) \left( \text{dx}(u_1) \ast \text{dx}(u_2) + \text{dy}(u_1) \ast \text{dy}(u_2) \right) + \text{on}(1, 2, 4, u_1=0) + \text{on}(3, u_1=1) ;
\]

Later variational forms can be used to construct right hand side vectors, matrices associated to them, or to define a new problem:

```plaintext
Xh u1, u2, v1, v2;
Mh p, q, ppp;
Xh bcl; bcl[] = a(0, Xh); // right hand side for boundary condition
Xh b;
```
matrix A = a(Xh, Xh, solver=CG); // the Laplace matrix
matrix<complex> CA = a(Xh, Xh, solver=CG); // the complex Laplace matrix

matrix Bx = bx(Xh, Mh); // Bx = (Bxij) and Bxij = bx(b^x_j, b^m_j)
// where b^x_j is a basis of Xh, and b^m_j is a basis of Mh.
matrix By = by(Xh, Mh); // By = (Byij) and Byij = by(b^x_j, b^m_j)

Note 16 The line of the matrix corresponding to test function on the bilinear form.

Note 17 The vector bc1[] contains the contribution of the boundary condition.

Here we have three matrices A, Bx, By, and we can solve the problem:
find u_1 \in X_h such that
\[ a(v_1, u_1) = b_y(v_1, f), \forall v_1 \in X_0_h, \]
u_1 = g, on \Gamma_1, and u_1 = 0 on \Gamma_1 \cup \Gamma_2 \cup \Gamma_4

with the following line (where f = x, and g = sin(x))

Mh f=x;
Xh g=sin(x);
b[] = Bx'*f[];
b[] += bc1[] .*bcx[]; // u_1= g on \Gamma_3 boundary see following remark
u1[] = A^-1*b[]; // solve the linear system

Note 18 The boundary condition is implemented by penalization and the vector bc1[] contains the contribution of the boundary condition u_1 = 1, so to change the boundary condition, we have just to multiply the vector bc1[] by the value f of the new boundary condition term by term with the operator .* The Section 7.6.2 StokesUzawa.edp gives a real example of using all this features.

We add automatic expression optimization by default, if this optimization trap you can remove the use of this optimization by writing for example :

varf a(u1,u2)= int2d(Th, optimize=false)(( dx(u1) * dx(u2) + dy(u1) * dy(u2) )
+ on(1,2,4, u1=0) + on(3, u1=1) ;

Remark, it is all possible to build interpolation matrix, like in the following exemple:

mesh TH = square(3,4);
mesh th = square(2,3);
mesh Th = square(4,4);

fespace VH(TH,P1);
fespace Vh(th,P1);
fespace Wh(Th,P1);

matrix B= interpolate(VH,Vh); // build interpolation matrix Vh->VH
matrix BB= interpolate(Wh,Vh); // build interpolation matrix Vh->Wh
and after some operation on sparce matrix are avialable for example

```cpp
int N=10;
real [int,int] A(N,N); // a full matrix
real [int] a(N),b(N);
A =0;
for (int i=0;i<N;i++)
{
  A(i,i)=1+i;
  if(i+1 < N) A(i,i+1)=-i;
  a[i]=i;
}
b=A*b;
cout << "xxxx\n";
matrix sparseA=A;
cout << sparseA << endl;
sparseA = 2*sparseA+sparseA';
sparseA = 4*sparseA+sparseA*5; //
matrix sparseB=sparseA+sparseA+sparseA; 
cout << "sparseB = " << sparseB(0,0) << endl;
```

4.10 Interpolation matrix

This possible to store the matrix of a linear interpolation operator from a finite element space $V_h$ to $W_h$ with interpolate function. Note by default the continuous finite function are extended. by continuity on ouside domain part.
The named parameter of function interpolate are:

- **inside**= set to true value to set the outside extention to zero.
- **t**= set to true value to get the transposed matrix
- **op**= set a int value $i$ to get

  0 the default value and interpolate of the function
  1 interpolate the $\partial_x$
  2 interpolate the $\partial_y$

A sample example mat_interpol.edp:

```cpp
mesh Th=square(4,4);
mesh Th4=square(2,2,[x*0.5,y*0.5]);
plot(Th,Th4,ps="ThTh4.eps",wait=1);
fespace Vh(Th,P1); fespace Vh4(Th4,P1);
fespace Wh(Th,P0); fespace Wh4(Th4,P0);

matrix IV= interpolate(Vh,Vh4); // here the function is
// extended by continuity
cout << " IV Vh<-Vh4 " << IV << endl;
```
4.11 Finite elements connectivity

With the following expression we can get the connectivity information of a finite element space $W_h$

- $W_h.n_{nt}$ gives the number of element of $W_h$
- $W_h.n_{dof}$ gives the number of degree of freedom or unknowns
- $W_h.n_{dofK}$ gives the number of degree of freedom on one element
- $W_h(k,i)$ gives the number of $i$th degree of freedom of element $k$.

See the following exemple:

```plaintext
mesh Th=square(5,5);
fespace Wh(Th,P2);
cout << " nb of degre of freedom : " << Wh.n_dof << endl;
cout << " nb of degre of freedom / ELEMENT : " << Wh.n_dofK << endl;
int k=2; // element 2
int kdf= Wh.n_dofK ;
cout << " df of element " << k << ":" ;
for (int i=0;i<kdf;i++)
  cout << Wh(k,i) << ":" ;
cout << endl;
```

and the output is:

```
Nb Of Nodes = 121
Nb of DF = 121
FESpace:Gibbs: old skyline = 5841 new skyline = 1377
nb of degre of freedom : 121
nb of degre of freedom / ELEMENT : 6
df of element 2:78 95 83 87 79 92
```
Chapter 5

Visualization

Numerical results in FEM create huge data, so it is very important to make obtained results visible. There are two ways of visualization in freefem++: One is default view supporting the draw of meshes, isovalue of real FE-functions and vector fields by the command plot (see Section 5.1). For documentation, freefem++ make the plotting stored as postscript files.

Another method is to use the external tools, for example, gnuplot (see Section 5.2), medit (see Section 5.3) using the command system.

5.1 Plot

With the command plot, meshes, isovalues and vector fields can be displayed. The parameters of the plot command can be , meshes, real FE functions , arrays of 2 real FE functions, arrays of two arrays of double, to plot respectively mesh, isovalue, vector field, or curve defined by the two arrays of double.

The named parameter are

wait= boolean expression to wait or not (by default no wait). If true we wait for a keyboard up event or mouse event, they respond to an event by the following characters

+ to zoom in around the mouse cursor,
- to zoom out around the mouse cursor,
= to restore de initial graphics state,
c to decrease the vector arrow coef,
C to increase the vector arrow coef,
r to refresh the graphic window,
f to toggle the filling between isovalues,
b to toggle the black and white,
g to toggle to grey or color ,
v to toggle the plotting of value,
p to save to a postscript file,
? to show all actives keyboard char,
to redraw, otherwise we continue.

\textbf{ps=} string expression to save the plot on postscript file

\textbf{coef=} the vector arrow coef between arrow unit and domain unit.

\textbf{fill=} to fill between isovalues.

\textbf{cmm=} string expression to write in the graphic window

\textbf{value=} to plot the value of isoline and the value of vector arrow.

\textbf{aspectratio=} boolean to be sure that the aspect ratio of plot is preserved or not.

\textbf{bb=} array of 2 array (like \([0.1,0.2], [0.5,0.6]\)), to set the bounding box and specify a partial view where the box defined by the two corner points [0.1,0.2] and [0.5,0.6].

\textbf{nbiso=} (int) sets the number of isovalues (20 by default)

\textbf{nbarrow=} (int) sets the number of colors of arrow values (20 by default)

\textbf{viso=} sets the array value of isovalues (an array real[int])

\textbf{varrow=} sets the array value of color arrows (an array real[int])

\textbf{bw=} (bool) sets or not the plot in black and white color.

\textbf{grey=} (bool) sets or not the plot in grey color.

For example:

\texttt{real[int] xx(10),yy(10);}
\texttt{mesh Th=square(5,5);}
\texttt{fespace Vh(Th,P1);}
\texttt{Vh uh=x*x+y*y,vh=-y^2+x^2;}
\texttt{int i;}
\texttt{// compute a cut}
\texttt{for (i=0;i<10;i++)}
\texttt{\{}
\texttt{x=i/10.; y=i/10.;}
\texttt{xx[i]=i;}
\texttt{yy[i]=uh; // value of uh at point (i/10. , i/10.)}
\texttt{\}}
\texttt{plot(Th,uh,[uh,vh],value=true,ps="three.eps",wait=true); // figure 5.1}
\texttt{// zoom on box defined by the two corner points [0.1,0.2] and [0.5,0.6]}
\texttt{plot(uh,[uh,vh],bb=[[0.1,0.2],[0.5,0.6]]},
\texttt{wait=true,gray=1,fill=1,value=1,ps="threeg.eps"); // figure 5.2}
\texttt{plot([xx,yy],ps="likegnu.eps",wait=true); // figure 5.3}
5.2 LINK WITH GNUPLOT

First this work only if gnuplot\footnote{http://www.gnuplot.info/} is installed, and only on unix computer. You just and to the previous example:

\begin{verbatim}
// file for gnuplot
{
    ofstream gnu("plot.gp");
    for (int i=0;i<=n;i++)
    {
        gnu << xx[i] << " " << yy[i] << endl;
    }
}
// the file plot.gp is close because the variable gnu is delete
// to call gnuplot command and wait 5 second (thanks to unix command)
// and make postscript plot
exec("echo 'plot "plot.gp" w l \
pause 5 \nset term postscript \nset output "gnuplot.eps" \nreplot \nquit' | gnuplot");
\end{verbatim}

5.3 link with medit

First this work only if medit\footnote{http://www-rocq.inria.fr/gamma/medit/medit.html} software is installed.

\begin{verbatim}
mesh Th=square(10,10,[2*x-1,2*y-1]);
\end{verbatim}
Figure 5.3: Plots a cut of uh. Note that a refinement of the same can be obtained in combination with gnuplot

Figure 5.4: Plots a cut of uh with gnuplot

```cpp
fespace Vh(Th,P1);
Vh u=2-x*x-y*y;
savemesh(Th,"mm",[x,y,u*.5]); // save mm.points and mm.faces file
  // for medit
  // build a mm.bb file

{ ofstream file("mm.bb");
  file << "2 1 1 " << u[].n << " 2 \n";
  int j;
  for (j=0; j<u[].n; j++)
    file << u[][j] << endl;
}
  // call medit command
exec("medit mm"); // clean files on unix OS
exec("rm mm.bb mm.faces mm.points");
```
Figure 5.5: medit plot
Chapter 6
Algorithms

The associated example is fully defined in algo.edp file.

6.1 conjugate Gradient/GMRES

If we want to solve the Euler problem: find \( x \in \mathbb{R}^n \) such that

\[
\frac{\partial J}{\partial x_i}(x) = 0
\]

where \( J \) is a functional (to minimize for example) from \( \mathbb{R}^n \) to \( \mathbb{R} \).

if the function is convex we can use the conjugate gradient to solve the problem, and we just need the function (named \( dJ \) for example) which compute \( \frac{\partial J}{\partial x_i} \), so the two parameters are the name of the function with prototype \( \text{func real[int]} \ dJ(\text{real[int]} \ & xx) \) which compute \( \frac{\partial J}{\partial x_i} \), a vector \( x \) of type \( \text{real[int]} \) to initialize the process and get the result.

**Note 19** You can use the macro tools (see 7.11) to build easily the differential, see example Newtow.edp.

Three versions are available:

**LinearCG** linear case , the functional \( J \) is quadratic \( J = \frac{1}{2}(x, Ax) - (b, x) \) where \( A \) is a symetric positive matrix .

**LinearCMRES** linear case ,where the functional \( J = \frac{1}{2}(x, Ax) - (b, x) \) where \( A \) is a matrix.

**NLCG** non linear case (the functional is just convex).

The named parameter of these three functions are:

**nbiter=** set the number of iteration (by default 100)

**precon=** set the preconditionner function (\( P \) for example) by default it is the identity, remark the prototype is \( \text{func real[int]} \ P(\text{real[int]} \ & xx) \).

**eps=** set the value of the stop test \( \varepsilon \ (= 10^{-6} \) by default) if positive then relative test \( \|dJ(x)\|_P \leq \varepsilon \ast \|dJ(x_0)\|_P \), otherwise the absolute test is \( \|dJ(x)\|_P \leq |\varepsilon| \).

77
\texttt{veps= set and return the value of the stop test, if positive then relative test } \|dJ(x)\|_P \leq \varepsilon \|dJ(x_0)\|_P, \text{ otherwise the absolute test is } ||dJ(X)||_P^2 \leq \varepsilon. \text{ The return value is minus the real stop test (remark: it is useful in loop).}

Example of use:

\begin{verbatim}
real[int] matx(10), b(10), x(10);

func real J(real[int] & x)
{
    real s=0;
    for (int i=0; i<x.n; i++)
        s += (i+1) * x[i] * x[i] * 0.5 - b[i] * x[i];
    return s;
}

// the grad of J (this is a affine version (the RHS is in )
func real[int] dJ(real[int] &x)
{
    for (int i=0; i<x.n; i++)
        matx[i]=(i+1) * x[i];
    matx -= b;
    return matx;
}

// the grad of the bilinear part of J (the RHS in remove)
func real[int] dJ0(real[int] &x)
{
    for (int i=0; i<x.n; i++)
        matx[i]=(i+1) * x[i];
    return matx;
}

//
func real error(real[int] & x, real[int] & b)
{
    real s=0;
    for (int i=0; i<x.n; i++)
        s += abs((i+1) * x[i] - b[i]);
    return s;
}

func real[int] matId(real[int] &x) { return x; }

b=1; x=0;    // here not rhs the CG (in dJ)
LinearCG(dJ,x,eps=1.e-6,nbiter=20,precon=matId);
cout << "LinearCG (Affine) : J(x) = " << J(x) << " err=" << error(x,b) << endl;

b=1; x=0;    // here rhs the CG (in not dJ0)
LinearCG(dJ0,x,b,eps=1.e-6,nbiter=20,precon=matId);
cout << "LinearCG (Linear) : J(x) = " << J(x) << " err=" << error(x,b) << endl;

b=1; x=0;    // here rhs the CG (in not dJ0)
LinearGMRES(dJ0,x,b,eps=1.e-6,nbiter=20,precon=matId);
cout << "LinearGMRES: J(x) = " << J(x) << " err=" << error(x,b) << endl;

b=1; x=0;    // set right hand side and initial gest
NLCG(dJ,x,eps=1.e-6,nbiter=20,precon=matId);
cout << "NLCG: J(x) = " << J(x) << " err=" << error(x,b) << endl;
\end{verbatim}
Two algorithms of COOOL, a package \[22\], are interfaced with the Newton Raphson method (call Newton) and the BFGS method. Be careful these algorithms, because the implementation use full matrix.

Example of utilization of algo.edp

```plaintext
func real J(real[int] & x)
{
    real s=0;
    for (int i=0;i<x.n;i++)
        s +=(i+1)*x[i]*x[i]*0.5 - b[i]*x[i];
    cout << "J="_<< s << " x="_<< x[0]_" << " "_<< x[1]_" << "...
";
    return s;
}
b=1; x=2; // set right hand side and initial guess
BFGS(J,dJ,x,eps=1.e-6,nbiter=20,nbiterline=20);
cout_"BFGS: J(x) = "_<< J(x)_" err="_<< error(x,b)_"_endl;
```
Chapter 7
Mathematical Models

7.1 Static Problems

7.1.1 Soap Film

Our starting point here will be the mathematical model to find the shape of soap film which is glued to the ring on the $xy$-plane

$$C = \{(x, y); \ x = \cos t, \ y = \sin t, \ 0 \leq t \leq 2\pi\}.$$

We assume the shape of the film is described as the graph $(x, y, u(x, y))$ of the vertical displacement $u(x, y) (x^2 + y^2 < 1)$ under a vertical pressure $p$ in terms of force per unit area and an initial tension $\mu$ in terms of force per unit length. Consider “small plane” ABCD, A:$(x, y, u(x,y))$, B:$(x, y, u(x+\delta x,y))$, C:$(x, y, u(x+\delta x,y+\delta y))$ and D:$(x, y, u(x,y+\delta y))$. Let us denote by $n(x,y) = (n_x(x,y), n_y(x,y), n_z(x,y))$ the normal vector of the surface $z = u(x, y)$. We see that the vertical force due to the tension $\mu$ acting along the edge AD is $-\mu n_x(x,y) \delta y$ and the vertical force acting along the edge AD is

$$\mu n_x(x+\delta x,y) \delta y \simeq \mu \left(n_x(x,y) + \frac{\partial n_x}{\partial x} \delta x\right)(x,y) \delta y.$$

Similarly, for the edges AB and DC we have

$$-\mu n_y(x,y) \delta x, \quad \mu \left(n_y(x,y) + \frac{\partial n_y}{\partial y} \delta y\right)(x,y) \delta x.$$

The force in the vertical direction on the surface ABCD due to the tension $\mu$ is given by the summation

$$\mu \left(\frac{\partial n_x}{\partial x}\right) \delta x \delta y + \mu \left(\frac{\partial n_y}{\partial y}\right) \delta y \delta x.$$
CHAPTER 7. MATHEMATICAL MODELS

Assuming small displacements, we have

$$\nu_x = \frac{\partial u/\partial x}{\sqrt{1 + (\partial u/\partial x)^2 + (\partial u/\partial y)^2}} \approx \partial u/\partial x,$$

$$\nu_y = \frac{\partial u/\partial y}{\sqrt{1 + (\partial u/\partial x)^2 + (\partial u/\partial y)^2}} \approx \partial u/\partial y.$$  

Letting $\delta x \to dx$, $\delta y \to dy$, we have the equilibrium of the virtual displacement of soap film on ABCD by $p$

$$\mu dx dy \partial^2 u/\partial x^2 + \mu dx dy \partial^2 u/\partial y^2 + pdxdy = 0.$$  

Using the Laplace operator $\Delta = \partial^2/\partial x^2 + \partial^2/\partial y^2$, we can find the virtual displacement write the following

$$-\Delta u = f \quad \text{in } \Omega \quad (7.1)$$

where $f = p/\mu$, $\Omega = \{(x, y); x^2 + y^2 < 1\}$. Poisson’s equation $f = \rho/\epsilon$ appear also in electrostatics taking the form of $f = \rho/\epsilon$ where $\rho$ is the charge density, $\epsilon$ the dielectric constant and $u$ is named as electrostatic potential. The soap film is glued to the ring $\partial \Omega = C$, then we have the boundary condition

$$u = 0 \quad \text{on } \partial \Omega \quad (7.2)$$

If the force is gravity, for simplify, we assume that $f = -1$.

Example 27 (a_tutorial.edp)

1: \texttt{border a(t=0,2*pi) \{ x = \cos(t); y = \sin(t); label=1; \}};
2: \texttt{mesh disk = buildmesh(a(50));}
3: \texttt{plot(disk);}
4: \texttt{fespace fempl(disk,P1);}
5: \texttt{fempl u,v;}
6: \texttt{func f = -1;}
7: \texttt{problem laplace(u,v) =}
8: \texttt{\quad int2d(disk)( \ dx(u)*dx(v) + dy(u)*dy(v) ) ; \quad // bilinear form}
9: \texttt{\quad - int2d(disk)( f*v ) ; \quad // linear form}
10: \texttt{\quad + on(1,u=0) ; \quad \quad // boundary condition}
11: \texttt{\quad func ue = (x^2+y^2-1)/4; \quad \quad // ue: exact solution}
12: \texttt{laplace;}
13: \texttt{fempl err = u - ue;}
14: \texttt{plot (u,ps="aTutorial.eps",value=true,wait=true);}
15: \texttt{plot (err,value=true,wait=true);}
16: \texttt{cout << "error L2=" << sqrt(int2d(disk)( err^2 ))<< endl;}
17: \texttt{cout << "error H10=" << sqrt( int2d(disk)((dx(u)-x/2)^2)+int2d(disk)((dy(u)-y/2)^2))<< endl;}
18: \texttt{disk = adaptmesh(disk,u,err=0.01);}
19: \texttt{plot(disk,wait=1);}
20: \texttt{laplace;}
21: \texttt{plot (u,value=true,wait=true);}
22: \texttt{err = u - ue; \quad \quad // become FE-function on adapted mesh}
23: \texttt{plot(err,value=true,wait=true);}
24: \texttt{cout << "error L2=" << sqrt(int2d(disk)( err^2 ))<< endl;}
7.1. STATIC PROBLEMS

Figure 7.1: isovalue of $u$

In 19th line, the $L^2$-error estimation between the exact solution $u_e$, 

$$
\|u_h - u_e\|_{0,\Omega} = \left( \int_{\Omega} |u_h - u_e|^2 \, dxdy \right)^{1/2}
$$

and from 20th line to 21th line, the $H^1$-error seminorm estimation

$$
|u_h - u_e|_{1,\Omega} = \left( \int_{\Omega} |\nabla u_h - \nabla u_e|^2 \, dxdy \right)^{1/2}
$$

are done on the initial mesh. The results are $\|u_h - u_e\|_{0,\Omega} = 0.000384045$, $|u_h - u_e|_{1,\Omega} = 0.0375506$. After the adaptation, we have $\|u_h - u_e\|_{0,\Omega} = 0.000109043$, $|u_h - u_e|_{1,\Omega} = 0.0188411$. So the numerical solution is improved by adaptation of mesh.

7.1.2 Electrostatics

We assume that there is no current and a time independent charge distribution. Then the electric field $E$ satisfy

$$
\text{div}E = \rho/\epsilon, \quad \text{curl}E = 0
$$

where $\rho$ is the charge density and $\epsilon$ is called the permittivity of free space. From the second equation in (7.3), we can introduce the electrostatic potential such that $E = -\nabla \phi$. Then we have Poisson equation $-\Delta \phi = f$, $f = -\rho/\epsilon$. We now obtain the equipotential line which is the level curve of $\phi$, when there are no charges except conductors $\{C_i\}_1^{...K}$. Let us assume $K$ conductors $C_1, \ldots, C_K$ within an enclosure $C_0$. Each one is held at an electrostatic potential $\phi_i$. We assume that the enclosure $C0$ is held at potential 0. In order to know $\phi(x)$ at any point $x$ of the domain $\Omega$, we must solve

$$
-\Delta \phi = 0 \quad \text{in} \ \Omega,
$$

Figure 7.2: a side view of $u$
where $\Omega$ is the interior of $C_0$ minus the conductors $C_i$, and $\Gamma$ is the boundary of $\Omega$, that is $\sum_{i=0}^{N} C_i$. Here $g$ is any function of $x$ equal to $\varphi_i$ on $C_i$ and to 0 on $C_0$. The second equation is a reduced form for:

$$\varphi = \varphi_i \text{ on } C_i, \ i = 1...N, \varphi = 0 \text{ on } C_0. \quad (7.5)$$

**Example 28** First we give the geometrical informations; $C_0 = \{(x,y); \ x^2 + y^2 = 5^2\}$, $C_1 = \{(x,y): \ 1/32(x-2)^2 + 1/32y^2 = 1\}$, $C_2 = \{(x,y): \ 1/32(x+2)^2 + 1/32y^2 = 1\}$. Let $\Omega$ be the disk enclosed by $C_0$ with the elliptical holes enclosed by $C_1$ and $C_2$. Note that $C_0$ is described counterclockwise, whereas the elliptical holes are described clockwise, because the boundary must be oriented so that the computational domain is to its left.

```plaintext
// a circle with center at (0,0) and radius 5
border C0(t=0,2*pi) { x = 5 * cos(t); y = 5 * sin(t); }
border C1(t=0,2*pi) { x = 2+0.3 * cos(t); y = 3 * sin(t); }
border C2(t=0,2*pi) { x = -2+0.3 * cos(t); y = 3 * sin(t); }
mesh Th = buildmesh(C0(60)+C1(-50)+C2(-50));

// figure 7.3
plot(Th,ps="electroMesh");

fespace Vh(Th,P1);
Vh uh,vh;

// unknown and test function.
problem Electro(uh,vh) =
  int2d(Th)( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) +
  on(C0,uh=0) +
  on(C1,uh=1) +
  on(C2,uh=-1) ;

// definition of the problem
// bilinear
// boundary condition on $C_0$
// +1 volt on $C_1$
// -1 volt on $C_2$

Electro;  // solve the problem, see figure 7.4 for the solution
plot(uh,ps="electro.eps",wait=true);  // figure 7.4
```

Figure 7.3: Disk with two elliptical holes  
Figure 7.4: Equipotential lines, where $C_1$ is located in right hand side
7.1.3 Aerodynamics

Let us consider a wing profile $S$ in a uniform flow. Infinity will be represented by a large circle $\Gamma_\infty$. As previously, we must solve

$$\Delta \varphi = 0 \quad \text{in} \quad \Omega, \quad \varphi|_S = c, \quad \varphi|_{\Gamma_\infty} = u_{\infty 1}x - u_{\infty 2}x \quad (7.6)$$

where $\Omega$ is the area occupied by the fluid, $u_\infty$ is the air speed at infinity, $c$ is a constant to be determined so that $\partial_n \varphi$ is continuous at the trailing edge $P$ of $S$ (so-called Kutta-Jukowski condition). Lift is proportional to $c$. To find $c$ we use a superposition method. As all equations in (7.6) are linear, the solution $\varphi_c$ is a linear function of $c$

$$\varphi_c = \varphi_0 + c\varphi_1, \quad (7.7)$$

where $\varphi_0$ is a solution of (7.6) with $c = 0$ and $\varphi_1$ is a solution with $c = 1$ and zero speed at infinity. With these two fields computed, we shall determine $c$ by requiring the continuity of $\partial \varphi/\partial n$ at the trailing edge. An equation for the upper surface of a NACA0012 (this is a classical wing profile in aerodynamics; the rear of the wing is called the trailing edge) is:

$$y := 0.17735\sqrt{x} - 0.075597x - 0.212836x^2 + 0.17363x^3 - 0.06254x^4. \quad (7.8)$$

Taking an incidence angle $\alpha$ such that $\tan \alpha = 0.1$, we must solve

$$-\Delta \varphi = 0 \quad \text{in} \quad \Omega, \quad \varphi|_{\Gamma_1} = y - 0.1x, \quad \varphi|_{\Gamma_2} = c, \quad (7.9)$$

where $\Gamma_2$ is the wing profile and $\Gamma_1$ is an approximation of infinity. One finds $c$ by solving:

$$\begin{align*}
-\Delta \varphi_0 &= 0 \quad \text{in} \quad \Omega, \quad \varphi_0|_{\Gamma_1} = y - 0.1x, \quad \varphi_0|_{\Gamma_2} = 0, \quad (7.10) \\
-\Delta \varphi_1 &= 0 \quad \text{in} \quad \Omega, \quad \varphi_1|_{\Gamma_1} = 0, \quad \varphi_1|_{\Gamma_2} = 1. \quad (7.11)
\end{align*}$$

The solution $\varphi = \varphi_0 + c\varphi_1$ allows us to find $c$ by writing that $\partial_n \varphi$ has no jump at the trailing edge $P = (1,0)$. We have $\partial_n \varphi - (\varphi(P^+) - \varphi(P))/\delta$ where $P^+$ is the point just above $P$ in the direction normal to the profile at a distance $\delta$. Thus the jump of $\partial_n \varphi$ is $(\varphi_0|_{P^+} + c(\varphi_1|_{P^+} - 1)) + (\varphi_0|_{P^-} + c(\varphi_1|_{P^-} - 1))$ divided by $\delta$ because the normal changes sign between the lower and upper surfaces. Thus

$$c = -\frac{\varphi_0|_{P^+} + \varphi_0|_{P^-}}{(\varphi_1|_{P^+} + \varphi_1|_{P^-} - 2)}, \quad (7.12)$$

which can be programmed as:

$$c = -\frac{\varphi_0(0.99,0.01) + \varphi_0(0.99,-0.01)}{(\varphi_1(0.99,0.01) + \varphi_1(0.99,-0.01) - 2)}. \quad (7.13)$$

**Example 29** // Computation of the potential flow around a NACA0012 airfoil. // The method of decomposition is used to apply the Joukowski condition // The solution is seeked in the form psio + beta psil and beta is // adjusted so that the pressure is continuous at the trailing edge

```plaintext
border a(t=0,2*pi) { x=5*cos(t); y=5*sin(t); }; // approximates infinity
```
\textbf{CHAPTER 7. MATHEMATICAL MODELS}

\begin{verbatim}
border upper(t=0,1) { x = t; y = 0.17735*sqrt(t)-0.075597*t - 0.212836*(t^2)+0.17363*(t^3)-0.06254*(t^4); }
border lower(t=1,0) { x = t; y = -0.17735*sqrt(t)+0.075597*t + 0.212836*(t^2)-0.17363*(t^3)+0.06254*(t^4); }
border c(t=0,2*pi) { x=0.8*cos(t)+0.5; y=0.8*sin(t); }

wait = true;
mesh Zoom = buildmesh(c(30)+upper(35)+lower(35));
mesh Th = buildmesh(a(30)+upper(35)+lower(35));

fespace Vh(Th,P2); // P1 FE space
Vh psi0,psi1,vh; // unknown and test function.
fespace ZVh(Zoom,P2);

solve Joukowski0(psi0,vh) = // definition of the problem
  int2d(Th)( dx(psi0)*dx(vh) + dy(psi0)*dy(vh) ) // bilinear form
  + on(a,psi0=y-0.1*x) // boundary condition form
  + on(upper,lower,psi0=0);
plot(psi0);

solve Joukowski1(psi1,vh) = // definition of the problem
  int2d(Th)( dx(psi1)*dx(vh) + dy(psi1)*dy(vh) ) // bilinear form
  + on(a,psi1=0) // boundary condition form
  + on(upper,lower,psi1=1);
plot(psi1);

// continuity of pressure at trailing edge
real beta = psi0(0.99,0.01)+psi0(0.99,-0.01);
beta = -beta / (psi1(0.99,0.01)+ psi1(0.99,-0.01)-2);

Vh psi = beta*psi1+psi0;
plot(psi);
ZVh Zpsi=psi;
plot(Zpsi,bw=true);
Vh cp = -dx(psi)^2 - dy(psi)^2;
plot(cp);
ZVh Zcp=cp;
plot(Zcp,nbiso=40);
\end{verbatim}

\subsection{Error estimation}

There are famous estimation between the numerical result $u_h$ and the exact solution $u$ of the problem \[1.1\] and \[1.2\]. If triangulations $\{T_h\}_{h \downarrow 0}$ is regular (see Section \[3.4\]), then we have the estimates

\[
\begin{align*}
|\nabla u - \nabla u_h|_{0,\Omega} & \leq C_1 h \quad (7.14) \\
\|u - u_h\|_{0,\Omega} & \leq C_2 h^2 \quad (7.15)
\end{align*}
\]

with constants $C_1$, $C_2$ independent of $h$, if $u$ is in $H^2(\Omega)$. It is known that $u \in H^2(\Omega)$ if $\Omega$ is convex.
In this section we check (7.14) and (7.15). We will pick up numericall error if we use the numerical derivative, so we will use the following for (7.14).

\[
\int_{\Omega} |\nabla u - \nabla u_h|^2 \, dx \, dy = \int_{\Omega} \nabla u \cdot \nabla (u - 2u_h) \, dx \, dy + \int_{\Omega} \nabla u_h \cdot \nabla u_h \, dx \, dy \\
= \int_{\Omega} f(u - 2u_h) \, dx \, dy + \int_{\Omega} f u_h \, dx \, dy
\]

The constants $C_1, C_2$ are depend on $T_h$ and $f$, so we will find them by freefem++. In general, we cannot get the solution $u$ as a elementary functions (see Section 2.6) even if spetical functions are added. Instead of the exact solution, here we use the approximate solution $u_0$ in $V_h(T_h, P_2)$, $h \sim 0$.

Example 30

```plaintext
1: mesh Th0 = square(100,100);
2: fespace V0h(Th0,P2);
3: V0h u0,v0;
4: func f = x*y;  // sin(pi*x)*cos(pi*y);
5:
6: solve Poisson0(u0,v0) = // bilinear form
7: int2d(Th0)( dx(u0)*dx(v0) + dy(u0)*dy(v0) )
8: - int2d(Th0)( f*v0 )  // linear form
9: + on(1,2,3,4,u0=0);  // boundary condition
10:
11: plot(u0);
12:
13: real[int] errL2(10), errH1(10);
14:
15: for (int i=1; i<=10; i++) {
16:    mesh Th = square(5+i*3,5+i*3);
17:    fespace Vh(Th,P1);
18:    fespace Ph(Th,P0);
19:    Ph h = hTriangle;  // get the size of all triangles
20:    Vh u,v;
21:    solve Poisson(u,v) =
```
We can guess that $C_1 = 0.0179253(0.0173266)$ and $C_2 = 0.0729566(0.0707543)$, where the numbers inside the parentheses are minimum in calculation.

### 7.1.5 Periodic

We now solve the Poisson equation

$$-\Delta u = \sin(x + \pi/4.) \times \cos(y + \pi/4.)$$

on the square $[0, 2\pi]^2$ under bi-periodic boundary condition $u(0, y) = u(2\pi, y)$ for all $y$ and $u(x, 0) = u(x, 2\pi)$ for all $x$. These boundary conditions are achieved from the definition of the periodic finite element space.

**Example 31 (periodic.edp)**

```plaintext
mesh Th=square(10,10,[2*x*pi,2*y*pi]);
  // defined the fespace with periodic condition
  // label : 2 and 4 are left and right side with y the curve abcissa
  // 1 and 2 are bottom and upper side with x the curve abcissa
fespace Vh(Th,P2,periodic=[[2,y],[4,y],[1,x],[3,x]]);
Vh uh,vh;  // unknown and test function.
func f=sin(x+pi/4.)*cos(y+pi/4.);  // right hand side function

problem laplace(uh,vh) =
  int2d(Th) ( dx(uh)*dx(vh) + dy(uh)*dy(vh) )
  + int2d(Th) ( -f*vh )
  ;
laplace;  // solve the problem plot(uh); // to see the result
plot(uh,ps="period.eps",value=true);
```

The periodic condition does not necessarily require parallel to the axis. Example 32 give such example.

**Example 32 (periodic4.edp)**

```plaintext
real r=0.25;  // a diamond with a hole

border a(t=0,1){x=-t+1; y=t;label=1;};
border b(t=0,1){ x=-t; y=1-t;label=2;};
border c(t=0,1){ x=t-1; y=-t;label=3;};
border d(t=0,1){ x=t; y=-1+t;label=4;};
```
Figure 7.7: The isovalue of solution $u$ with periodic boundary condition

```cpp
border e(t=0,2*pi)( x=r*cos(t); y=-r*sin(t); label=0;);
int n = 10;

mesh Th = buildmesh(a(n)+b(n)+c(n)+d(n)+e(n));
plot(Th,wait=1);
real r2=1.732;

func abs=sqrt(x^2+y^2);
// warning for periodic condition:
// side a and c
// on side a (label 1) x ∈ [0,1] or x - y ∈ [-1,1]
// on side c (label 3) x ∈ [-1,0] or x - y ∈ [-1,1]
// so the common abcissa can be repectively x and x + 1
// or you can can try curviline abcissa x-y and x-y
// 1 first way
// fespace Vh(Th,P2,periodic=[[2,1+x],[4,x],[1,x],[3,1+x]]);
// 2 second way
fespace Vh(Th,P2,periodic=[[2,x+y],[4,x+y],[1,x-y],[3,x-y]]);

Vh uh,vh;

func f=(y+x+1)*(y-x-1)*(y-x+1)*(y-x-1);
real intf = int2d(Th)(f);
real mTh = int2d(Th) (1);
real k = intf/mTh;
cout << k << endl;

problem laplace(uh,vh) =
    int2d(Th) ( dx(uh)*dx(vh) + dy(uh)*dy(vh) ) + int2d(Th) ( (k-f)*vh ) ;
laplace;
plot(uh,wait=1,ps="perio4.eps");
```
7.1.6 Poisson with mixed boundary condition

Here we consider the Poisson equation with mixed boundary value problems: For given functions $f$ and $g$, find $u$ such that

$$
-\Delta u = f \quad \text{in } \Omega \\
u = g \quad \text{on } \Gamma_D, \quad \partial u / \partial n = 0 \quad \text{on } \Gamma_N
$$

where $\Gamma_D$ is a part of the boundary $\Gamma$ and $\Gamma_N = \Gamma \setminus \Gamma_D$. The solution $u$ has the singularity at the points $\{\gamma_1, \gamma_2\} = \Gamma_D \cap \Gamma_N$. When $\Omega = \{(x,y); -1 < x < 1, 0 < y < 1\}$, $\Gamma_N = \{(x,y); -1 \leq x < 0, y = 0\}$, $\Gamma_D = \partial \Omega \setminus \Gamma_N$, the singularity will appear at $\gamma_1 = (0,0)$, $\gamma_2 = (-1,0)$, and $u$ has the expression

$$u = K_i u_S + u_R, \quad u_R \in H^2(\text{near } \gamma_i), \quad i = 1, 2$$

with a constants $K_i$. Here $u_S = r_j^{1/2} \sin(\theta_j/2)$ by the local polar coordinate $(r_j, \theta_j$ at $\gamma_j$ such that $(r_1, \theta_1) = (r, \theta)$. Instead of polar coordinate system $(r, \theta)$, we use that $r = \sqrt{x^2 + y^2}$ and $\theta = \arctan(y/x)$ in FreeFem++.

Example 33 Assume that $f = -2 \times 30(x^2 + y^2)$ and $g = u_e = 10(x^2 + y^2)^{1/4} \sin(\tan^{-1}(y/x)/2) + 30(x^2y^2)$, where $u_e$ is the exact solution.

```plaintext
1: border N(t=0,1) { x=-1+t; y=0; label=1; }
2: border D1(t=0,1) { x=t; y=0; label=2; }
3: border D2(t=0,1) { x=1; y=t; label=2; }
4: border D3(t=0,2) { x=1-t; y=1; label=2; }
5: border D4(t=0,1) { x=-1; y=1-t; label=2; }
6:
7: mesh T0h = buildmesh(N(10)+D1(10)+D2(10)+D3(20)+D4(10));
8: plot(T0h, wait=true);
9: fespace V0h(T0h,P1);
10: V0h u0, v0;
11:
```
7.1. STATIC PROBLEMS

f = -2 * 30 * (x^2 + y^2);                   // given function
the singular term of the solution is K * us (K: constant)
us = sin(atan2(y, x)/2) * sqrt( sqrt(x^2 + y^2) );
K = 10.;
ue = K * us + 30 * (x^2 * y^2);

solve Poisson0(u0, v0) =
    int2d(T0h) ( dx(u0) * dx(v0) + dy(u0) * dy(v0) )  // bilinear form
- int2d(T0h) ( f * v0 )  // linear form
+ on(2, u0 = ue) ;     // boundary condition

mesh Th = adaptmesh(T0h, us);
for (int i = 0; i < 5; i++)
    mesh Th = adaptmesh(Th, us);

fespace Vh(Th, P1);
Vh u, v;
solve Poisson(u, v) =
    int2d(Th) ( dx(u) * dx(v) + dy(u) * dy(v) )  // bilinear form
- int2d(Th) ( f * v )  // linear form
+ on(2, u = ue) ;     // boundary condition

/* plot the solution */
plot(Th, ps = "adaptDNmix.ps");
plot(u, wait = true);

H1e = sqrt( int2d(Th) ( dx(uue) * 2 + dy(uue) * 2 + uue^2 ) );

/* calculate the H1 Sobolev norm */
Vh err0 = u0 - ue;
Vh err = u - ue;
Vh H1err0 = int2d(Th) ( dx(err0)^2 + dy(err0)^2 + err0^2 ) ;
Vh H1err = int2d(Th) ( dx(err)^2 + dy(err)^2 + err^2 ) ;
cout << "Relative error in first mesh " << int2d(Th) (H1err0)/H1e<<endl;
cout << "Relative error in adaptive mesh " << int2d(Th) (H1err)/H1e<<endl;

From 24th line to 28th, adaptation of meshes are done using the base of singular term. In 42th line, H1e = \|u_e\|_{1, \Omega} is calculated. In last 2 lines, the relative errors are calculated, that is,

\|u_h - u_e\|_{1, \Omega}/H1e = 0.120421
\|u_h - u_e\|_{1, \Omega}/H1e = 0.0150581

where u_h^0 is the numerical solution in T0h and u_h is u in this program.
7.1.7 Adaptation with residual error indicator

We do metric mesh adaption and compute the classical residual error indicator $\eta_T$ on the element $T$ for the Poisson problem.

Example 34 (adaptindicatorP2.edp)  
First, we solve the same problem as in a previous example.

```plaintext
1: border ba(t=0,1.0) {x=t; y=0; label=1;};  // see Fig.3.13
2: border bb(t=0,0.5) {x=1; y=t; label=2;};
3: border bc(t=0,0.5) {x=1-t; y=0.5; label=3;};
4: border bd(t=0.5,1) {x=0.5; y=t; label=4;};
5: border be(t=0.5,1) {x=1-t; y=1; label=5;};
6: border bf(t=0.0,1) {x=0; y=1-t; label=6;};
7: mesh Th = buildmesh (ba(6) + bb(4) + bc(4) + bd(4) + be(4) + bf(6));
8: save mesh (Th,"th.msh");
9: fespace Vh(Th,P2);
10: fespace Nh(Th,P0);
11: Vh u,v;
12: Nh rho;
13: real[int] viso(21);
14: for (int i=0;i<viso.n;i++)
15: viso[i]=10.^(+(i-16.)/2.);
16: real error=0.01;
17: func f=(x-y);
18: problem Probem1(u,v,solver=CG,eps=1.0e-6) =
19: int2d(Th,qforder=5)( u*v*1.0e-10 + dx(u)*dx(v) + dy(u)*dy(v))
20: + int2d(Th,qforder=5)( -f*v);
21: /*************
Now, the local error indicator $\eta_T$ is:

$$\eta_T = \left( h_T^2 ||f + \Delta u_h||_{L^2(T)}^2 + \sum_{e \in \mathcal{E}_K} h_e \left[ \frac{\partial u_h}{\partial n_e} \right]^2_{L^2(e)} \right)^{\frac{1}{2}}$$

where $h_T$ is the longest’s edge of $T$, $\mathcal{E}_K$ is the set of $T$ edge not on $\Gamma = \partial \Omega$, $n_T$ is the outside unit normal to $K$, $h_e$ is the length of edge $e$, $[g]$ is the jump of the function $g$ across edge (left value minus right value).

Of course, we can use a variational form to compute $\eta^2_T$, with test function constant function in each triangle.

29: **************/
30:
31: varf indicator2(uu,chiK) =
32: intalledges(Th)(chiK*lenEdge*square(jump(N.x*dx(u)+N.y*dy(u))))
33: +int2d(Th)(chiK*square(hTriangle*(f+dxx(u)+dyy(u))))
34: for (int i=0;i< 4;i++)
35: {
36: Probem1;
37: cout << uu[].min << " " << uu[].max << endl;
38: plot(u,wait=1);
39: cout << " indicator2 " << endl;
40: 
41: rho[] = indicator2(0,Nh);
```
7.2 Elasticity

Consider an elastic plate with undeformed shape $\Omega \times [0, h]$ in $\mathbb{R}^3$, $\Omega \subset \mathbb{R}^2$. By the deformation of the plate, we assume that a point $P(x_1, x_2, x_3)$ moves to $P(\xi_1, \xi_2, \xi_3)$. The vector $u = (u_1, u_2, u_3) = (\xi_1 - x_1, \xi_2 - x_2, \xi_3 - x_3)$ is called displacement vector. By the deformation, the line segment $x, x + \tau \Delta x$ moves approximately to $x + u(x), x + \tau \Delta x + u(x + \tau \Delta x)$ for small $\tau$, where $x = (x_1, x_2, x_3), \Delta x = (\Delta x_1, \Delta x_2, \Delta x_3)$. We now calculate the ratio between two segments

$$\eta(\tau) = \tau^{-1} |\Delta x|^{-1} \left( |u(x + \tau \Delta x) - u(x)| + |\tau \Delta x| - \tau |\Delta x| \right)$$

then we have (see e.g. [11, p.32])

$$\lim_{\tau \to 0} \eta(\tau) = (1 + 2 e_{ij} \nu_i \nu_j)^{1/2} - 1, \quad 2 e_{ij} = \frac{\partial u_k}{\partial x_i} \frac{\partial u_k}{\partial x_j} + \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$
The cylinder is assumed to be very thin and subjected to no load on the ends
\[ \varepsilon_{ij}(u) = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \]
and the following is called \textit{small} strain tensor.
The tensor \( e_{ij} \) is called \textit{finite strain tensor}.
Consider the small plane \( \Delta \Pi(x) \) centered at \( x \) with the unit normal direction \( n = (n_1, n_2, n_3) \), then the surface on \( \Delta \Pi(x) \) at \( x \) is
\[ (\sigma_{1j}(x)n_j, \sigma_{2j}(x)n_j, \sigma_{3j}(x)n_j) \]
where \( \sigma_{ij}(x) \) is called \textit{stress tensor} at \( x \). Hooke’s law is the assumption of a linear relation between \( \sigma_{ij} \) and \( \varepsilon_{ij} \) such as
\[ \sigma_{ij}(x) = c_{ijkl}(x)e_{ij}(x) \]
with the symmetry \( c_{ijkl} = c_{jikl} = c_{ijlk} = c_{klij} \).
If Hooke’s tensor \( c_{ijkl}(x) \) do not depend on the choice of coordinate system, the material is called \textit{isotropic} at \( x \). If \( c_{ijkl} \) is constant, the material is called \textit{homogeneous}. In homogeneous isotropic case, there is \textit{Lamé constants} \( \lambda, \mu \) (see e.g. [11, p.43]) satisfying
\[ \sigma_{ij} = \lambda \delta_{ij} \text{div} u + 2 \mu \varepsilon_{ij} \tag{7.17} \]
where \( \delta_{ij} \) is Kronecker’s delta. We assume that the elastic plate is fixed on \( \Gamma_D \times ]-h, h[ \), \( \Gamma_D \subset \partial \Omega \). If the body force \( f = (f_1, f_2, f_3) \) is given in \( \Omega \times ]-h, h[ \) and surface force \( g \) is given in \( \Gamma_N \times ]-h, h[ \), \( \Gamma_N = \partial \Omega \setminus \Gamma_D \), then the equation of equilibrium is given as follows:
\[ \begin{align*}
- \partial_j \sigma_{ij} &= f_i \text{ in } \Omega \times ]-h, h[, \quad i = 1, 2, 3 \\
\sigma_{ij} n_j &= g_i \text{ on } \Gamma_N \times ]-h, h[, \quad u_i = 0 \text{ on } \Gamma_D \times ]-h, h[, \quad i = 1, 2, 3
\end{align*} \tag{7.18} \tag{7.19} \]
We now explain the plain elasticity.

\textbf{Plain strain:} On the end of plate, the contact condition \( u_3 = 0, g_3 = 0 \) is satisfied. In this case, we can suppose that \( f_3 = g_3 = u_3 = 0 \) and \( u(x_1, x_2, x_3) = \overline{u}(x_1, x_2) \) for all \( -h < x_3 < h \).

\textbf{Plain stress:} The cylinder is assumed to be very thin and subjected to no load on the ends \( x_3 = \pm h \), that is,
\[ \sigma_{3i} = 0, \quad x_3 = \pm h, \quad i = 1, 2, 3 \]
The assumption leads that \( \sigma_{3i} = 0 \) in \( \Omega \times ]-h, h[ \) and \( u(x_1, x_2, x_3) = \overline{u}(x_1, x_2) \) for all \( -h < x_3 < h \).

\textbf{Generalized plain stress:} The cylinder is subjected to no load on the ends \( x_3 = \pm h \). Introducing the mean values with respect to thickness,
\[ \overline{u}_i(x_1, x_2) = \frac{1}{2h} \int_{-h}^{h} u(x_1, x_2, x_3)dx_3 \]
and we derinde \( \overline{u}_3 \equiv 0 \). Similary we define the mean values \( \overline{f}, \overline{g} \) of the body force and surface force as well as the mean values \( \tau_{ij} \) and \( \sigma_{ij} \) of the components of stress and strain, respectively.
In what follows we omit the overlines of $\bar{u}$, $\bar{f}$, $\bar{g}$, $\bar{\varepsilon}_{ij}$ and $\bar{\varepsilon}_{ij}$. Then we obtain similar equation of equilibrium given in (7.18) replacing $\Omega \times [-h, h]$ with $\Omega$ and changing $i = 1, 2$. In the case of plane stress, $\sigma_{ij} = \lambda^* \delta_{ij} \text{div} u + 2\mu \varepsilon_{ij}$, $\lambda^* = (2\lambda\mu) / (\lambda + \mu)$.

The equations of elasticity are naturally written in variational form for the displacement vector $u(x) \in V$ as

$$
\int_{\Omega} [2\mu \varepsilon_{ij}(u)\varepsilon_{ij}(v) + \lambda \varepsilon_{ii}(u)\varepsilon_{jj}(v)] = \int_{\Omega} f \cdot v + \int_{\Gamma} g \cdot v, \forall v \in V
$$

where $V$ is the linear closed subspace of $H^1(\Omega)^2$.

**Example 35 (Beam.edp)** Consider elastic plate with the undeformed rectangle shape $[0, 10] \times [0, 2]$. The body force is the gravity force $f$ and the boundary force $g$ is zero on lower and upper side. On the two vertical sides of the beam are fixed.

```plaintext
int bottombeam = 2;
border a(t=2,0) { x=0; y=t; label=1;}; // left beam
border b(t=0,10) { x=t; y=0; label=bottombeam;}; // bottom of beam
border c(t=0,2) { x=10; y=t; label=1;}; // right beam
border d(t=0,10) { x=10-t; y=2; label=3;}; // top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th, [P1,P1]);
Vh [uu,vv], [w,s];
cout << "lambda,mu,gravity =" << lambda << " " << mu << " " << gravity << endl;

#define 2d( th )
   2*mu*(dx(uu)*dx(w)+dy(vv)*dy(s)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/2 )
   + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))

solve bb([[uu,vv],[w,s]]) =
   int2d(th)( 2*mu*(dx(uu)*dx(w)+dy(vv)*dy(s)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/2 )
   + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))
   )
   + int2d(th) (-gravity*s)
   + on(1,uu=0, vv=0);

plot([uu,vv], wait=1);
plot([uu,vv], wait=1, bb=[[-0.5,2.5],[2.5,-0.5]]);
mesh th1 = movemesh(th, [x+uu, y+vv]);
plot(th1, wait=1);
```
7.2.1 Fracture Mechanics

Consider the plate with the crack whose undeformed shape is a curve $\Sigma$ with the two edges $\gamma_1$, $\gamma_2$. We assume the stress tensor $\sigma_{ij}$ is the state of plate stress regarding $(x, y) \in \Omega_{\Sigma} = \Omega \setminus \Sigma$. Here $\Omega$ stands for the undeformed shape of elastic plate without crack. If the part $\Gamma_N$ of the boundary $\partial \Omega$ is fixed and a load $L = (f, g) \in L^2(\Omega)^2 \times L^2(\Gamma_N)^2$ is given, then the displacement $u$ is the minimizer of the potential energy functional 

$$
\mathcal{E}(v; L, \Omega_{\Sigma}) = \int_{\Omega_{\Sigma}} \{w(x, v) - f \cdot v\} - \int_{\Gamma_N} g \cdot v
$$

over the functional space $V(\Omega_{\Sigma})$,

$$
V(\Omega_{\Sigma}) = \{v \in H^1(\Omega_{\Sigma})^2; \; v = 0 \; \text{ on } \Gamma_D = \partial \Omega \setminus \Gamma_N\},
$$

where $w(x, v) = \sigma_{ij}(v)\varepsilon_{ij}(v)/2$,

$$
\sigma_{ij}(v) = C_{ijkl}(x)\varepsilon_{kl}(v), \quad \varepsilon_{ij}(v) = \left(\partial v_i/\partial x_j + \partial v_j/\partial x_i\right)/2, \quad (C_{ijkl} : \text{ Hooke’s tensor}).
$$

If the elasticity is homogeneous isotropic, then the displacement $u(x)$ is decomposed in an open neighborhood $U_k$ of $\gamma_k$ as in (see e.g. [12])

$$
u(x) = \sum_{l=1}^{2} K_i(\gamma_k)\tau_k^{1/2} S_{kl}^C(\theta_k) + u_{k,R}(x) \quad \text{ for } x \in \Omega_{\Sigma} \cap U_k, \; k = 1, 2 \quad (7.20)
$$

with $u_{k,R} \in H^2(\Omega_{\Sigma} \cap U_k)^2$, where $U_k$, $k = 1, 2$ are open neighborhoods of $\gamma_k$ such that $\partial L \cap U_1 = \gamma_1$, $\partial L \cap U_2 = \gamma_2$, and

$$
S_{k1}^C(\theta_k) = \frac{1}{4\mu (2\pi)^{1/2}} \begin{bmatrix}
\frac{2\kappa - 1}{\cos(\theta_k/2)} - \cos(3\theta_k/2) \\
-\frac{2\kappa + 1}{\sin(\theta_k/2) + \sin(3\theta_k/2)}
\end{bmatrix},
$$

$$
S_{k2}^C(\theta_k) = \frac{1}{4\mu (2\pi)^{1/2}} \begin{bmatrix}
-\frac{2\kappa - 1}{\sin(\theta_k/2) + 3\sin(3\theta_k/2)} \\
-\frac{2\kappa + 1}{\cos(\theta_k/2) + \cos(3\theta_k/2)}
\end{bmatrix},
$$

where $\mu$ is the shear modulus of elasticity, $\kappa = 3 - 4\nu$ ($\nu$ is the Poisson’s ratio) for plane strain and $\kappa = \frac{3\nu}{1-\nu}$ for plane stress.

The coefficients $K_1(\gamma_i)$ and $K_2(\gamma_i)$, which are important parameters in fracture mechanics, are called stress intensity factors of the opening mode (mode I) and the sliding mode (mode II), respectively.

For simplicity, we consider the following simple crack

$$
\Omega = \{(x, y) : -1 < x < 1, -1 < y < 1\}, \quad \Sigma = \{(x, y) : -1 \leq x \leq 0, y = 0\}
$$

with only one crack tip $\gamma = (0, 0)$. Unfortunately, freefem++ cannot treat crack, so we use the modification of the domain with U-shape channel (see Fig. 3.26) with $d = 0.0001$. The undeformed crack $\Sigma$ is approximated by

$$
\Sigma_d = \{(x, y) : -1 \leq x \leq -10 \ast d, -d \leq y \leq d\} \\
\cup \{(x, y) : -10 \ast d \leq x \leq 0, -d + 0.1 \ast x \leq y \leq d - 0.1 \ast x\}
$$

and $\Gamma_D = \emptyset$ in Fig. 3.26. In this example, we use three technique:
7.2. ELASTICITY

- Fast Finite Element Interpolator from the mesh $\mathcal{Th}$ to zoom for the scale-up of near $\gamma$.
- After obtaining the displacement vector $\mathbf{u} = (u, v)$, we shall watch the deformation of the crack near $\gamma$ as follows,

  $$\text{mesh Plate} = \text{movemesh}(\text{Zoom}, [x+u, y+v]);$$
  $$\text{plot(Plate);}$$

- Important technique is adaptive mesh, because the large singularity occur at $\gamma$ as shown in (7.20).

First example create mode I deformation by the opposed surface force on $B$ and $T$ in the vertical direction of $\Sigma$, and the displacement is fixed on $R$.
In a laboratory, fracture engineer use photoelasticity to make stress field visible, which shows the principal stress difference

$$\sigma_1 - \sigma_2 = \sqrt{(\sigma_{11} - \sigma_{22})^2 + 4\sigma_{12}^2}$$  \hspace{1cm} (7.22)

where $\sigma_1$ and $\sigma_2$ are the principal stresses. In opening mode, the photoelasticity make symmetric pattern concentrated at $\gamma$.

Example 36 (Crack Opening, $K_2(\gamma) = 0$) {CrackOpen.edp}

```plaintext
real d = 0.0001;
int n = 5;
real cb=1, ca=1, tip=0.0;
border L1(t=0,ca-d) { x=-cb; y=-d-t; }
border L2(t=0,ca-d) { x=-cb; y=ca-t; }
border B(t=0,2) { x=cb*(t-1); y=-ca; }
border C1(t=0,1) { x=-ca*(1-t)+(tip-10*d)*t; y=d; }
border C21(t=0,1) { x=(tip-10*d)*(1-t)+tip*t; y=d*(1-t); }
border C22(t=0,1) { x=(tip-10*d)*t+tip*(1-t); y=-d*t; }
border C3(t=0,1) { x=(tip-10*d)*(1-t)-ca*t; y=-d; }
border C4(t=0,2*d) { x=-ca; y=-d*t; }
border R(t=0,2) { x=cb; y=cb*(t-1); }
border T(t=0,2) { x=cb*(1-t); y=ca; }
mesh Th = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)+C21(3)+C22(3)+C3(n)+R(n)+T(n));

cb=0.1; ca=0.1;
plot(Th,wait=1);

mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n)+C21(3)+C22(3)+C3(n)+R(n)+T(n));

plot(Zoom,wait=1);
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
fdomain Vh(Th,[P2,P2]);
fdomain zVh(Zoom,P2);
Vh [u,v], [w,s];
solve Problem([u,v],[w,s]) =
  int2d(Th) (2*mu*(dx(u)*dx(w)+(dx(v)+dy(u))*(dx(s)+dy(w)))/4 + lambda*(dx(u)+dy(v))*(dx(w)+dy(s))/2);
```
\(-\text{int}_1(\text{Th,T}) (0.1*(4-x)*s) + \text{int}_1(\text{Th,B}) (0.1*(4-x)*s)\) 
+ on(R,u=0) + on(R,v=0); // fixed

zVh Sx, Sy, Sxy, N;
for (int i=1; i<=5; i++) {
    mesh Plate = movemesh(Zoom,[x+u,y+v]); // deformation near \(\gamma\)
    Sx = lambda*(dx(u)+dy(v)) + 2*mu*dx(u);
    Sy = lambda*(dx(u)+dy(v)) + 2*mu*dy(v);
    Sxy = mu*(dy(u) + dx(v));
    N = 0.1*sqrt((Sx-Sy)^2 + 4*Sxy^2); // principal stress difference
    if (i==1) {
        plot(Plate,ps="1stCOD.eps",bw=1); // Fig. 7.10
        plot(N,ps="1stPhoto.eps",bw=1); // Fig. 7.10
    } else if (i==5) {
        plot(Plate,ps="LastCOD.eps",bw=1); // Fig. 7.11
        plot(N,ps="LastPhoto.eps",bw=1); // Fig. 7.11
        break;
    }
    Th=adaptmesh(Th,[u,v]);
    Problem;
}

Figure 7.10: Crack open displacement (COD) and Principal stress difference in the first mesh

Figure 7.11: COD and Principal stress difference in the last adaptive mesh

It is difficult to create mode II deformation by the opposed shear force on \(\mathfrak{B}\) and \(\tau\) that is observed in a laboratory. So we use the body shear force along \(\Sigma\), that is, the \(x\)-component \(f_1\) of the body force \(f\) is given by

\[
f_1(x,y) = H(y - 0.001) \ast H(0.1 - y) - H(-y - 0.001) \ast H(y + 0.1)
\]

where \(H(t) = 1\) if \(t > 0\); \(= 0\) if \(t < 0\).

Example 37 (Crack Sliding, \(K_2(\gamma) = 0\)) (use the same mesh Th)
bcb=0.01; cab=0.01;
mesh Zoom = buildmesh (L1(n/2)+L2(n/2)+B(n)+C1(n) 
+ C21(3)+C22(3)+C3(n)+R(n)+T(n));
(use same FE-space Vh and elastic modulus)
fespace Vh1(Th,P1);
7.3. NONLINEAR STATIC PROBLEMS

Vh 1  \( f_x = \left( (y > 0.001) \times (y < 0.1) \right) - \left( (y < -0.001) \times (y > -0.1) \right) \)

\[ \text{solve } \text{Problem}([u,v],[w,s]) = \]
\[ \text{int2d}(\Theta) \{
2 \times \mu \times (dx(u) \times dx(w) + ((dx(v) + dy(u)) \times (dx(s) + dy(w))) / 4 ) \\
+ \lambda \times (dx(u) + dy(v)) \times (dx(w) + dy(s)) / 2
\}
\]
\[ \text{int2d}(\Theta) (f \times w) \\
+ \text{on}(R,u=0) + \text{on}(R,v=0); \quad \text{// fixed} \]

\[ \text{for (int } i=1; i<=3; i++) \{
\text{mesh Plate = movemesh(Zoom, [x+u, y+v]);} \quad \text{// deformation near } \gamma
\text{Sx = } \lambda \times (dx(u) + dy(v)) + 2 \times \mu \times dx(u); \quad \text{Sxy = } \mu \times (dy(u) + dx(v)); \quad \text{N = 0.1 \times 1 \times sqrt((Sx-Sy)^2+4 \times Sxy^2);} \quad \text{// principal stress difference}
\text{if (i==1) } \{
\text{plot(Plate, ps= ”1stCOD2.eps”, bw=1);} \quad \text{// Fig. 7.13}
\text{plot(N, ps= ”1stPhoto2.eps”, bw=1);} \quad \text{// Fig. 7.12}
\text{else if (i==3) } \{
\text{plot(Plate, ps= ”LastCOD2.eps”, bw=1)}; \quad \text{// Fig. 7.13}
\text{plot(N, ps= ”LastPhoto2.eps”, bw=1);} \quad \text{// Fig. 7.13}
\text{break;}
\}
\text{Th=adaptmesh(Th, [u, v]);}
\text{Problem;}
\}

Figure 7.12: (COD) and Principal stress difference in the first mesh

Figure 7.13: COD and Principal stress difference in the last adaptive mesh

7.3 Nonlinear Static Problems

We propose how to solve the following non-linear academic problem of minimization of a functional

\[ J(u) = \int_\Omega f(|\nabla u|^2) - u \ast b \]
where $u$ is function of $H^1_0(\Omega)$ and $f$ defined by

$$f(x) = a \cdot x + x - \ln(1 + x), \quad f'(x) = a + \frac{x}{1 + x}, \quad f''(x) = \frac{1}{(1 + x)^2}$$

### 7.3.1 Non linear conjugate gradient algorithm

```cpp
// mesh definition of \Omega
mesh Th=square(10,10); fespace Vh(Th,P1); // finite element space fespace Ph(Th,P0); // make optimization

A small hack to construct a function

$$C l = \begin{cases} 1 & \text{on interior degree of freedom} \\ 0 & \text{on boundary degree of freedom} \end{cases}$$

```cpp
// Hack to construct an array : // 1 on interior nodes and 0 on boundary nodes varf vCl(u,v) = on(1,2,3,4,u=1); Vh Cl; Cl[]= vCl(0,Vh,tgv=1); // 0 and tgv real tgv=Cl[].max; // Cl[] = -Cl[]; Cl[] += tgv; Cl[] /=tgv;
```cpp

the definition of $f$, $f'$, $f''$ and $b$

```cpp
// $J(u) = \int_{\Omega} f(\nabla u)^2 - \int \Omega ub
// f(x) = a \cdot x + x - \ln(1 + x), \quad f'(x) = a + \frac{x}{1 + x}, \quad f''(x) = \frac{1}{(1 + x)^2}$
real a=0.001;

func real f(real u) { return u*a+u-log(1+u); }
func real df(real u) { return a+u/(1+u); }
func real ddf(real u) { return 1/((1+u)*(1+u)); }
Vh b=1; // to defined b

// the routine to compute the functional $J$
func real J(real[iv] & x) {
    Vh u;u[]=x;
    real r=int2d(Th)(f( dx(u)*dx(u) + dy(u)*dy(u) ) - b*u) ;
    cout << "J(x) =" << r << " " << x.min << " " << x.max << end1;
    return r;
}
```

The function to compute $DJ$, where $u$ is the current solution.

```cpp
Vh u=0; // the current value of the solution Vh alpha; // of store $f(\|\nabla u\|²)$ int iter=0;
```
7.3. NONLINEAR STATIC PROBLEMS

\[ \alpha = \text{df}(\; \text{dx}(u) \times \text{dx}(u) + \text{dy}(u) \times \text{dy}(u) \;) \]  // optimization

```c
func real[int] dJ(real[int] & x)
{
    int verb=verbosity; verbosity=0;
    Vh u; u[] = x;
    alpha = df(\; \text{dx}(u) \times \text{dx}(u) + \text{dy}(u) \times \text{dy}(u) \;) \quad \text{// optimization}
    varf au(uh,vh) = \text{int2d}(\text{Th})(\; \alpha \times (\; \text{dx}(uh) \times \text{dx}(vh) + \text{dy}(uh) \times \text{dy}(vh) \;) - b \times vh); x= au(0,Vh);
    x = x \times \text{Cl}[]; \quad \text{// the grad in 0 on boundary}
    verbosity=verb;
    return x; \quad \text{// warning no return of local array variable}
}
```

We want to construct also a preconditionner function \( C \) with solving the problem: find \( u_h \in V_{0h} \) such that

\[ \forall v_h \in V_{0h}, \quad \int_{\Omega} \alpha \nabla u_h . \nabla v_h = \int_{\Omega} b v_h \]

where \( \alpha = f(|\nabla u|^2) \).

```c
varf alap(uh,vh,solver=Cholesky,init=iter)=
    \text{int2d}(\text{Th})(\; \alpha \times (\; \text{dx}(uh) \times \text{dx}(vh) + \text{dy}(uh) \times \text{dy}(vh) \;) \) + \text{on}(1,2,3,4,uh=0);
varf amass(uh,vh,solver=Cholesky,init=iter)= \text{int2d}(\text{Th})(\; uh \times vh \) + \text{on}(1,2,3,4,uh=0);
matrix Amass = alap(Vh,Vh,solver=CG);
matrix Alap = alap(Vh,Vh,solver=Cholesky,factorize=1); \quad \text{// the preconditionner function}
```

```c
func real[int] C(real[int] & x)
{
    real[int] u(x.n);
    u = Amass \times x;
    x = Alap^\text{-1} \times u;
    x = x \times \text{Cl}[];
    return x; \quad \text{// no return of local array variable}
}
```

A good idea to solve the problem is make 10 iteration of the conjugate gradient, recompute the preconditioning and restart the conjugate gradient:

```c
verbosity=5;
int conv=0;
real eps=1e-6;
for (int i=0;i<20;i++)
{
    conv = NLCG(dJ,u[],nbiter=10,precon=C,veps=eps);
    if (conv) break; \quad \text{// if converge break loop}
    alpha = df(\; \text{dx}(u) \times \text{dx}(u) + \text{dy}(u) \times \text{dy}(u) \;) \quad \text{// recompute alpha optimization}
    Alap = alap(Vh,Vh,solver=Cholesky,factorize=1);
}
```

7.3.2 Newton Raphson algorithm

Now, we solve the Euler problem $\nabla J(u) = 0$ with Newton Raphson algorithm, that is,

$$u^{n+1} = u^n - (\nabla^2 J(u^n))^{-1} * dJ(u^n)$$

First we introduce the two variational form $vdJ$ and $vhJ$ to compute respectively $\nabla J$ and $\nabla^2 J$

```plaintext
//methode of Newton Raphson to solve dJ(u)=0;

// $u^{n+1} = u^n - (\frac{\partial dJ}{\partial u_i})^{-1} * dJ(u^n)$

Ph dalpha; // to store $f''(\|\nabla u\|^2)$ optimisation
```

```plaintext
// the variational form of evaluate dJ = $\nabla J$

// ---------------------------------------------
varf vdJ(uh,vh) = int2d(Th)( alpha*( dx(uh)\*dx(vh) + dy(uh)\*dy(vh) ) - b\*vh)
+ on(1,2,3,4, uh=0);

// the variational form of evaluate ddJ = $\nabla^2 J$

// hJ(uh,vh) = f''(u)* ( dx(uh)\*dx(vh) + dy(uh)\*dy(vh)
// + f''(u)\* ( dx(uh)\*dx(uh) + dy(uh)\*dy(uh) ) * ( dx(uh)\*dx(vh) + dy(uh)\*dy(vh) ) )

varf vhJ(uh,vh) = int2d(Th)( alpha* ( dx(uh)\*dx(vh) + dy(uh)\*dy(vh) )
+ dalpha*( dx(uh)\*dx(vh) + dy(uh)\*dy(vh) )\* ( dx(uh)\*dx(uh) + dy(uh)\*dy(uh) ) )
+ on(1,2,3,4, uh=0);

// the Newton algorithm
Vh v,w;
u=0;
for (int i=0;i<100;i++)
{
    alpha = df( dx(u)\*dx(u) + dy(u)\*dy(u) ) ; // optimization
dalpha = ddf( dx(u)\*dx(u) + dy(u)\*dy(u) ) ; // optimization
    v[]= vdJ(0,Vh); // $v = \nabla J(u)$
    real res= v[]\’*v[];
    cout << i << " residu^2 = " << res << endl;
}
```
7.4 **Eigenvalue Problems**

This section depends on your FreeFem++ compilation process (see README.arpack), of this tools. This tool is available in FreeFem++ if the word “eigenvalue” appear in line “Load:”, like:

```plaintext
-- FreeFem++ v1.28 (date Thu Dec 26 10:56:34 CET 2002)
file: LapEigenValue.edp
Load: lg_fem lg_mesh eigenvalue
```

This tool is based on the arpack++ the object-oriented version of ARPACK eigenvalue package [1].

The function EigenValue compute the generalized eigenvalue of $A u = \lambda B u$ where $\sigma = \sigma$ is the shift of the method. The matrix $OP$ is defined with $A - \sigma B$. The return value is the number of converged eigenvalue (can be greater than the number of eigenvalue $nev$):

```plaintext
int k=EigenValue(OP,B,nev= , sigma= );
```

where the matrix $OP = A - \sigma B$ with a solver and boundary condition, and the matrix $B$.

- **sym** the problem is symmetric (all the eigen value are real)
- **nev** the number desired eigenvalues (nev) close to the shift.
- **value** the array to store the real part of the eigenvalues
- **ivalue** the array to store the imag. part of the eigenvalues
- **vector** the array to store the eigenvectors. For real nonsymmetric problems, complex eigenvectors are given as two consecutive vectors, so if eigenvalue $k$ and $k + 1$ are complex conjugate eigenvalues, the $k$th vector will contain the real part and the $k + 1$th vector the imaginary part of the corresponding complex conjugate eigenvectors.
- **tol** the relative accuracy to which eigenvalues are to be determined;
- **sigma** the shift value;
- **maxit** the maximum number of iterations allowed;
- **ncv** the number of Arnoldi vectors generated at each iteration of ARPACK.

Example 38 (lapEigenValue.edp) In the first example, we compute the eigenvalue and the eigenvector of the Dirichlet problem on square \( \Omega = [0, \pi]^2 \).

The problem is find: \( \lambda \), and \( \nabla u_\lambda \) in \( \mathbb{R} \times H^1_0(\Omega) \)

\[
\int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega uv \quad \forall v \in H^1_0(\Omega)
\]

The exact eigenvalues are \( \lambda_{n,m} = (n^2 + m^2) \), \((n, m) \in \mathbb{N}^2\) with the associated eigenvectors are \( u_{m,n} = \sin(nx) \times \sin(my) \).

We use the generalized inverse shift mode of the arpack++ library, to find 20 eigenvalue and eigenvector close to the shift value \( \sigma = 20 \).

```
// Computation of the eigen value and eigen vector of the
// Dirichlet problem on square [0, \pi]^2
// ----------------------------------------
// we use the inverse shift mode
// the shift is given with the real sigma
// -------------------------------------
// find \( \lambda \) and \( u_\lambda \in H^1_0(\Omega) \) such that:
// \[
// \int_\Omega \nabla u_\lambda \nabla v = \lambda \int_\Omega u_\lambda v, \forall v \in H^1_0(\Omega)
// \]
verbosity=10;

mesh Th=square(20,20,[pi*x,pi*y]);

fespace Vh(Th,P2);
Vh u1,u2;

real sigma = 20; // value of the shift

// OP = A - sigma B ; // the shifted matrix
varf op(u1,u2)= int2d(Th)( dx(u1)*dx(u2) + dy(u1)*dy(u2) - sigma*u1*u2 )
+ on(1,2,3,4,u1=0) ; // Boundary condition

varf b([u1],[u2]) = int2d(Th)( u1*u2 ) ; // no Boundary condition

matrix OP= op(Vh,Vh,solver=Crout,factorize=1); // crout solver because the
matrix in not positive
matrix B= b(Vh,Vh,solver=CG,eps=1e-20);

int nev=20; // number of computed eigen value close to sigma

real[int] ev(nev); // to store the nev eigenvalue
Vh[int] eV(nev); // to store the nev eigenvector

int k=EigenValue(OP,B,sym=true,sigma=sigma,value=ev,vector=eV,
    tol=1e-10,maxit=0,ncv=0);
```
7.4. EIGENVALUE PROBLEMS

```c
for (int i=0; i<k; i++)
{
    ul=eV[i];
    real gg = int2d(Th)(dx(ul)*dx(ul) + dy(ul)*dy(ul));
    real mm= int2d(Th)(ul*ul) ;
    cout << " ---- " << i<< " " << ev[i]<< " err= "
        <<int2d(Th)(dx(ul)*dx(ul) + dy(ul)*dy(ul) - (ev[i])*ul*ul) << " --- "<<endl;
    plot(eV[i],cmm="Eigen Vector "+i+" valeur =" + ev[i] ,wait=1,value=1);
}
```

The output of this example is:

```
Nb of edges on Mortars = 0
Nb of edges on Boundary = 80, neb = 80
Nb Of Nodes = 1681
Nb of DF = 1681
Real symmetric eigenvalue problem: A*x - B*x*lambda

Thanks to ARPACK++ class ARrcSymGenEig
Real symmetric eigenvalue problem: A*x - B*x*lambda
Shift and invert mode sigma=20

Dimension of the system : 1681
Number of 'requested' eigenvalues : 20
Number of 'converged' eigenvalues : 20
Number of Arnoldi vectors generated: 41
Number of iterations taken : 2

Eigenvalues:
lambda[1]: 5.0002
lambda[2]: 8.00074
lambda[3]: 10.0011
lambda[4]: 10.0011
lambda[5]: 13.002
lambda[6]: 13.0039
lambda[7]: 17.0046
lambda[8]: 17.0048
lambda[9]: 18.0083
lambda[10]: 20.0096
lambda[11]: 20.0096
lambda[12]: 25.014
lambda[13]: 25.0283
lambda[14]: 26.0159
lambda[15]: 26.0159
lambda[16]: 29.0258
lambda[17]: 29.0273
lambda[18]: 32.0449
lambda[19]: 34.049
```
\[ \lambda_{20} = 34.0492 \]

--- 0 5.0002 err= -0.000225891 ---
--- 1 8.00074 err= -0.000787446 ---
--- 2 10.0011 err= -0.00134596 ---
--- 3 10.0011 err= -0.00134619 ---
--- 4 13.002 err= -0.00227747 ---
--- 5 13.0039 err= -0.004179 ---
--- 6 17.0046 err= -0.00623649 ---
--- 7 17.0048 err= -0.00639952 ---
--- 8 18.0083 err= -0.00862954 ---
--- 9 20.0096 err= -0.0110483 ---
--- 10 20.0096 err= -0.0110696 ---
--- 11 25.014 err= -0.0154412 ---
--- 12 25.0283 err= -0.0291014 ---
--- 13 26.0159 err= -0.0218532 ---
--- 14 26.0159 err= -0.0218544 ---
--- 15 29.0258 err= -0.0311961 ---
--- 16 29.0273 err= -0.0326472 ---
--- 17 32.0449 err= -0.0457328 ---
--- 18 34.049 err= -0.0530978 ---
--- 19 34.0492 err= -0.0536275 ---

Figure 7.14: Isovalue of 11th eigenvector 
\[ u_{4,3} - u_{3,4} \]

Figure 7.15: Isovalue of 12th eigenvector 
\[ u_{4,3} + u_{3,4} \]
7.5 Evolution Problems

\texttt{freefem++} also solve evolution problems such as the heat problem

\[ \frac{\partial u}{\partial t} - \mu \Delta u = f \quad \text{in } \Omega \times [0, T[ , \quad (7.23) \]

\[ u(x, 0) = u_0(x) \quad \text{in } \Omega ; \quad (\partial u/\partial n)(x, t) = 0 \quad \text{on } \partial \Omega \times [0, T[ . \]

with a positive viscosity coefficient \( \mu \) and homogeneous Neumann boundary conditions. We solve (7.23) by FEM in space and finite differences in time. We use the definition of the partial derivative of the solution in the time derivative,

\[ \frac{\partial u}{\partial t}(x, y, t) = \lim_{\tau \to 0} \frac{u(x, y, t) - u(x, y, t - \tau)}{\tau} \]

which indicate that \( u^m(x, y) = u(x, y, m\tau) \) imply

\[ \frac{\partial u}{\partial t}(x, y, m\tau) \simeq \frac{u^m(x, y) - u^{m-1}(x, y)}{\tau} \]

The time descrezation of heat equation (7.24) is as follows:

\[ \frac{u^{m+1} - u^m}{\tau} - \mu \Delta u^{m+1} = f^{m+1} \quad \text{in } \Omega \]

\[ u^0(x) = u_0(x) \quad \text{in } \Omega ; \quad \partial u^{m+1}/\partial n(x) = 0 \quad \text{on } \partial \Omega , \quad \text{for all } m = 0, \cdots , [T/\tau] , \]

which is so-called \textit{backward Euler method} for (7.24). Multiplying the test function \( v \) both sides of the formula just above, we have

\[ \int_\Omega \{ u^{m+1}v - \tau \Delta u^{m+1}v \} = \int_\Omega \{ u^m + \tau f^{m+1} \} v . \]

By the divergence theorem, we have

\[ \int_\Omega \{ u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v \} - \int_{\partial \Omega} \tau (\partial u^{m+1}/\partial n) v = \int_\Omega \{ u^m v + \tau f^{m+1} \} v . \]

By the boundary condition \( \partial u^{m+1}/\partial n = 0 \), it follows that

\[ \int_\Omega \{ u^{m+1}v + \tau \nabla u^{m+1} \cdot \nabla v \} - \int_\Omega \{ u^m v + \tau f^{m+1} \} v = 0 . \quad (7.25) \]

Using the identity just above, we can calculate the finite element approximation \( u^m_h \) of \( u^m \) in a step-by-step manner with respect to \( t \).

\textbf{Example 39} We now solve the following example with the exact solution \( u(x, y, t) = tx^4 \).

\[ \frac{\partial u}{\partial t} - \mu \Delta u = x^4 - \mu 12tx^2 \quad \text{in } \Omega \times [0, 3[, \quad \Omega = ]0, 1[^2 \]

\[ u(x, y, 0) = 0 \quad \text{on } \Omega , \quad u|_{\partial \Omega} = t \ast x^4 \]
// heat equation \partial_t u = -\mu \Delta u = x^4 - \mu 12tx^2

mesh Th=square(16,16);
fe space Vh(Th,P1);

Vh u,v,uu,f,g;
real dt = 0.1, mu = 0.01;
problem dHeat(u,v) =
    int2d(Th) ( u*v + dt*mu*(dx(u)*dx(v) + dy(u)*dy(v)) )
    + int2d(Th) (- uu*v - dt*f*v )
    + on(1,2,3,4,u=g);
real t = 0; // start from t=0
uu = 0; // u(x,y,0)=0
for (int m=0;m<=3/dt;m++)
{
    t=t+dt;
    f = x^4-mu * t * 12 * x^2;
    g = t * x^4;
    dHeat;
    plot(u,wait=true);
    uu = u;
    cout <<"t="<<t<<"\text{L}^2\text{-Error}="<<sqrt( int2d(Th) ((u-t*x^4) )^2 ) << endl;
}

In the last statement, the $L^2$-error $\left(\int_\Omega |u - tx^4|^2\right)^{1/2}$ is calculated at $t = m\tau$, $\tau = 0.1$. At $t = 0.1$, the error is 0.000213269. The errors increase with $m$ and 0.00628589 at $t = 3$.
The iteration of the backward Euler (7.25) is made by for loop (see Section 2.8).

Note 21 The stiffness matrix in loop is used over and over again. freefem++ support reuses of stiffness matrix.

7.5.1 Mathematical Theory on time difference

In this section, we show the advantage of the backward Euler. Let $V, H$ be separable Hilbert space and $V$ is dense in $H$. Let $a$ be a continuous bilinear form over $V \times V$ with coercivity and symmetry. Then $\sqrt{a(v,v)}$ become equivalent to the norm $\|v\|$ of $V$.

problem Ev(f, Omega): For a given $f \in L^2(0,T;V')$, $u^0 \in H$

\[
\frac{d}{dt}(u(t),v) + a(u(t),v) = (f(t),v) \quad \forall v \in V, \quad a.e. t \in [0,T] \quad (7.26)
\]

\[
u(0) = u^0
\]

where $V'$ is the dual space of $V$. Then, there is an unique solution $u \in L^\infty(0,T;H) \cap L^2(0,T;V)$.

Let us denote the time step by $\tau > 0$, $N_T = [T/\tau]$. For the discretization, we put $u^n = u(n\tau)$ and consider the time difference for each $\theta \in [0,1]$}

\[
\frac{1}{\tau} (u^{n+1}_h - u^n_h, \phi_i) + a (u^{n+\theta}_h, \phi_i) = (f^{n+\theta}, \phi_i) \quad (7.27)
\]

\[
u^{n+\theta}_h = \theta u^{n+1}_h + (1-\theta)u^n_h, \quad f^{n+\theta} = \theta f^{n+1} + (1-\theta)f^n
\]
7.5. EVOLUTION PROBLEMS

Formula (7.27) is the forward Euler scheme if $\theta = 0$, Crank-Nicolson scheme if $\theta = 1/2$, the backward Euler scheme if $\theta = 1$.

Unknown vectors $u^n = (u^n_1, \ldots, u^n_M)^T$ in

$$u^n_h(x) = u^n_1 \phi_1(x) + \cdots + u^n_m \phi_m(x), \quad u^n_1, \ldots, u^n_m \in \mathbb{R}$$

are obtained from solving the matrix

$$(M + \theta \tau A) u^{n+1} = \{M - (1 - \theta) \tau A\} u^n + \tau \{\theta f^{n+1} + (1 - \theta) f^n\}$$

(7.28)

$$M = (m_{ij}), \quad m_{ij} = (\phi_j, \phi_i), \quad A = (a_{ij}), \quad a_{ij} = a(\phi_j, \phi_i)$$

Refer [17, pp.70–75] for solvability of (7.28). The stability of (7.28) is in [17, Theorem 2.13]:

Let $\{T_h\}_{h>0}$ be regular triangulations (see Section 3.4). Then there is a number $c_0 > 0$ independent of $h$ such that,

$$|u^n_h|^2 \leq \begin{cases} 
\frac{1}{\delta} \left\{ |u^0_h|^2 + \tau \sum_{k=0}^{n-1} \|f^{k+\theta}\|_{V^*_h}^2 \right\} & \theta \in [0, 1/2) \\
|u^0_h|^2 + \tau \sum_{k=0}^{n-1} \|f^{k+\theta}\|_{V^*_h}^2 & \theta \in [1/2, 1]
\end{cases}$$

(7.29)

if the following are satisfied:

1. When $\theta \in [0, 1/2)$, then we can take a time step $\tau$ in such a way that

$$\tau < \frac{2(1-\delta)}{(1-2\theta)c_0^2} h$$

(7.30)

for arbitrary $\delta \in (0, 1)$.

2. When $1/2 \leq \theta \leq 1$, we can take $\tau$ arbitrary.

Example 40

```c
mesh Th=square(12,12);
fespace Vh(Th,P1);
fespace Ph(Th,P0);
Ph h = hTriangle; // mesh sizes for each triangle
real tau = 0.1, theta=0.;
func real f(real t) {
    return x^2*(x-1)^2 + t*(-2 + 12*x - 11*x^2 - 2*x^3 + x^4);
}
ofstream out("err02.csv"); // file to store calculations
out << "mesh size = "<<h[].max<<", time step = "<<tau<<endl;
for (int n=0;n<5/tau;n++) \n    out<<n*tau<<endl;
Vh u,v,oldU;
Vh f1, f0;
problem aTau(u,v) =
    int2d(Th) ( u*v + theta*tau*(dx(u)*dx(v) + dy(u)*dy(v) + u*v)) -
    int2d(Th) (oldU*v - (1-theta)*tau*(dx(oldU)*dx(v)+dy(oldU)*dy(v)+oldU*v)) -
    int2d(Th) (tau*( theta*f1+(1-theta)*f0 )*v )
;```
while (theta <= 1.0) {
    real t = 0, T=3; // from t=0 to T
    oldU = 0; // u(x,y,0)=0
    out <<theta<<",";
    for (int n=0;n<T/tau;n++) {
        t = t+tau;
        f0 = f(n*tau); f1 = f((n+1)*tau);
        aTau;
        oldU = u;
        plot(u);
        Vh uex = t*x^2*(1-x)^2; // exact sol. = tr^2(1−x)^2
        Vh err = u - uex; // err=FE-sol - exact
        out<< abs(err[].max)/abs(uex[].max) <<","; // ∥err∥_{L^∞(Ω)}/∥u_{ex}∥_{L^∞(Ω)}
    }
    out << endl;
    theta = theta + 0.1;
}

Figure 7.16: max_{x∈Ω} |u^n_h(θ) − u_{ex}(nτ)|/ max_{x∈Ω} |u_{ex}(nτ)| at n = 0,1,…,29

We can see in Fig. 7.16 that u^n_h(θ) become unstable at θ = 0.4, and figures are omitted in the case θ < 0.4.

7.5.2 Convection

The hyperbolic equation

$$\partial_t u - \alpha \cdot \nabla u = f; \quad \text{for a vector-valued function } \alpha, \quad \partial_t = \frac{\partial}{\partial t},$$

(7.31)

appear frequently in scientific problems, for example, Navier-Stokes equation, Convection-Diffusion equation, etc.

In the case of 1-dimensional space, we can easily find the general solution (x,t) ↦ u(x,t) = u^0(x − at) of the following equation, if α is constant,

$$\partial_t u + \alpha \partial_x u = 0, \quad u(x,0) = u^0(x),$$

(7.32)
because \( \partial_t u + \alpha \partial_x u = -\alpha \dot{u} + a \dot{u} = 0 \), where \( \dot{u} = du^0(x)/dx \). Even if \( \alpha \) is not constant construction, the principle is similar. One begins the ordinary differentielle equation (with convention which \( \alpha \) is prolonged by zero apart from \( (0, L) \times (0, T) \)): 
\[
\dot{X}(\tau) = -\alpha(X(\tau), \tau), \quad \tau \in (0, t) \quad X(t) = x
\]

In this equation \( \tau \) is the variable and \( x, t \) is parameters, and we denote the solution by \( X_{x,t}(\tau) \). Then it is noticed that \( (x, t) \to u(X(\tau), \tau) \) in \( \tau = t \) satisfy the equation
\[
\partial_t v + \alpha \partial_x v = \partial_t X \dot{v} + a \partial_x X \dot{v} = 0
\]

and by the definition \( \partial_t X = \dot{X} = -\alpha \) and \( \partial_x X = \partial_x x \) in \( \tau = t \), because if \( \tau = t \) we have \( X(\tau) = x \). The general solution of (7.32) is thus the value of the boundary condition in \( X_{x,t}(0) \), it is has to say \( u(x, t) = u^0(X_{x,t}(0)) \) if \( X_{x,t}(0) \) is on the \( x \) axis, \( u(x, t) = u^0(X_{x,t}(0)) \) if \( X_{x,t}(0) \) is on the axis of \( t \).

In higher dimension \( \Omega \subset R^d \), \( d = 2, 3 \), the equation of the convection is written
\[
\partial_t u + \mathbf{a} \cdot \nabla u = 0 \quad \text{in} \Omega \times (0, T)
\]

where \( \mathbf{a}(x, t) \in R^d \). \texttt{freefem++} implements the Characteristic-Galerkin method for convection operators. Recall that the equation [7.31] can be discretized as
\[
\frac{Du}{Dt} = f \quad \text{i.e.} \quad \frac{du}{dt}(X(t), t) = f(X(t), t) \quad \text{where} \quad \frac{dX}{dt}(t) = \alpha(X(t), t)
\]

where \( D \) is the total derivative operator. So a good scheme is one step of backward convection by the method of Characteristics-Galerkin

\[
\frac{1}{\tau} \left( u^{m+1}(x) - u^m(X^m(x)) \right) = f^m(x) \tag{7.33}
\]

where \( X^m(x) \) is an approximation of the solution at \( t = m\tau \) of the ordinary differential equation
\[
\frac{dX}{dt}(t) = \alpha^m(X(t)), \quad X((m+1)\tau) = x.
\]

where \( \alpha^m(x) = (\alpha_1(x, m\tau), \alpha_2(x, m\tau)) \). Because, by Taylor’s expansion, we have
\[
u^m(X(m\tau)) = u^m(X((m+1)\tau)) - \tau \sum_{i=1}^d \frac{\partial u^m}{\partial x_i}(X((m+1)\tau)) \frac{\partial X_i}{\partial t}((m+1)\tau) + o(\tau)
\]

where \( X_i(t) \) are the i-th component of \( X(t) \), \( u^m(x) = u(x, m\tau) \) and we used the chain rule and \( x = X((m+1)\tau) \). From (7.34), it follows that
\[
u^m(X^m(x)) = u^m(x) - \tau \alpha^m(x) \cdot \nabla u^m(x) + o(\tau). \tag{7.35}
\]

Also we apply Taylor’s expansion for \( t \to u^m(x - \alpha^m(x)t) \), \( 0 \leq t \leq \tau \), then
\[
u^m(x - \alpha t) = u^m(x) - \tau \alpha^m(x) \cdot \nabla u^m(x) + o(\tau).
\]
Putting
\[ \text{convect} (\alpha, \tau, u^m) = u^m (x - \alpha^m \tau), \]
we can get the approximation
\[ u^m (X^m(x)) \approx \text{convect} ([a_1^m, a_2^m], \tau, u^m) \]
by \( x = X((m + 1)\tau). \)

A classical convection problem is that of the “rotating bell” (quoted from [10][p.16]). Let \( \Omega \) be the unit disk centered at 0, with its center rotating with speed \( \alpha_1 = y, \alpha_2 = -x \). We consider the problem (7.31) with \( f = 0 \) and the initial condition \( u(x, 0) = u^0(x) \), that is, from (7.33)
\[ u^{m+1}(x) = u^m(X^m(x)) \approx \text{convect}(\alpha, \tau, u^m). \]

The exact solution is \( u(x, t) = u(X(t)) \) where \( X \) equals \( x \) rotated around the origin by an angle \( \theta = -t \) (rotate in clockwise). So, if \( u^0 \) in a 3D perspective looks like a bell, then \( u \) will have exactly the same shape, but rotated by the same amount. The program consists in solving the equation until \( T = 2\pi \), that is for a full revolution and to compare the final solution with the initial one; they should be equal.

**Example 41 (convect.edp)**

```plaintext
border C(t=0, 2*pi) { x=cos(t); y=sin(t); }; // the unit circle
mesh Th = buildmesh(C(70)); // triangulates the disk
fespace Vh(Th,P1);
Vh u0 = exp(-10*((x-0.3)^2 +(y-0.3)^2)); // give u^0

real dt = 0.17, t=0; // time step
Vh a1 = -y, a2 = x; // rotation velocity
Vh u; // u^m+1
for (int m=0; m<2*pi/dt; m++) {
  t += dt;
  u=convect([a1,a2],dt,u0); // u^{m+1} = u^m(X^m(x))
  u0=u; // m++
  plot(u,cmm="convection: t="+t + ", min="+ u[].min + ", max="+ u[].max,wait=0);
}
```

**Note 22** The scheme \text{convect} is unconditionally stable, then the bell become lower and lower (the maximum of \( u^{37} \) is 0.406 as shown in Fig. 7.18).

### 7.5.3 Two-dimensional Black-Scholes equation

In mathematical finance, an option on two assets is modeled by a Black-Scholes equations in two space variables, (see for example Wilmott’s book : a student introduction to mathematical finance, Cambridge University Press).

\[
\partial_t u + \frac{(\sigma_1 x)^2}{2} \partial^2 u \frac{\partial^2 u}{\partial x^2} + \frac{(\sigma_2 y)^2}{2} \partial^2 u \frac{\partial^2 u}{\partial y^2} + \rho xy \frac{\partial^2 u}{\partial x \partial y} + rS_1 \frac{\partial u}{\partial x} + rS_2 \frac{\partial u}{\partial y} - rP = 0
\]
which is to be integrated in \((0, T) \times \mathbb{R}^+ \times \mathbb{R}^+\) subject to, in the case of a put

\[
    u(x, y, T) = (K - \max(x, y))^+.
\]  

(7.37)

Boundary conditions for this problem may not be so easy to device. As in the one dimensional case the PDE contains boundary conditions on the axis \(x_1 = 0\) and on the axis \(x_2 = 0\), namely two one dimensional Black-Scholes equations driven respectively by the data \(u(0, +\infty, T)\) and \(u(+\infty, 0, T)\). These will be automatically accounted for because they are embedded in the PDE. So if we do nothing in the variational form (i.e. if we take a Neuman boundary condition at these two axis in the strong form) there will be no disturbance to these. At infinity in one of the variable, as in 1D, it makes sense to match the final condition:

\[
    u(x, y, t) \approx (K - \max(x, y))^+ e^{r(T-t)} \text{ when } |x| \to \infty
\]  

(7.38)

For an American put we will also have the constraint

\[
    u(x, y, t) \geq (K - \max(x, y))^+ e^{r(T-t)}.
\]  

(7.39)

We take

\[
    \sigma_1 = 0.3, \quad \sigma_2 = 0.3, \quad \rho = 0.3, \quad r = 0.05, \quad K = 40, \quad T = 0.5
\]  

(7.40)

An implicit Euler scheme with projection is used and a mesh adaptation is done every 10 time steps. The first order terms are treated by the Characteristic Galerkin method, which, roughly, approximates

\[
    \frac{\partial u}{\partial t} + a_1 \frac{\partial u}{\partial x} + a_2 \frac{\partial u}{\partial y} \approx \frac{1}{\tau} \left( u^{n+1}(x) - u^n(x - \alpha \tau) \right)
\]  

(7.41)
int L=80;
int LL=80;

border aa(t=0,L){x=t;y=0;};
border bb(t=0,LL){x=L;y=t;};
border cc(t=L,0){x=t ;y=LL;};
border dd(t=LL,0){x = 0; y = t;};

mesh th = buildmesh(aa(m)+bb(m)+cc(m)+dd(m));
fespace Vh(th,P1);

real sigmax=0.3;
real sigmay=0.3;
real rho=0.3;
real r=0.05;
real K=40;
real dt=0.01;
real eps=0.3;

func f = max(K-max(x,y),0.);

Vh u=f,v,w;

func beta = 1; // (w<=f-eps)*eps + (w>=f) + 
( w<f)*(w-f-eps)*(eps+(w-f+eps)/eps)*(1-eps);

plot(u,wait=1);

th = adaptmesh(th,u,abserror=1,nbjacoby=2,
err=0.004, nbvex=5000, omega=1.8,ratio=1.8, nbsmooth=3,
splitpbedge=1, maxsubdiv=5,rescaling=1 );

u=u;

Vh xveloc = -x*r+x*sxmax^2+x*rho*sigmax*sigmay/2;
Vh yveloc = -y*r+y*sigmay^2+y*rho*sigmax*sigmay/2;

int j=0;
int n;

problem eq1(u,v,init=j, solver=LU) = int2d(th) ( 
    u*v*(r+1/dt/beta) + dx(u)*dx(v)*x*sigmax^2/2. + dy(u)*dy(v)*y*sigmay^2/2. + 
    dx(u)*dy(v)*rho*sigmax*sigmay*x*y/2. + dx(u)*dy(v)*rho*sigmax*sigmay*x*y/2. 
    + int2d(th)( -v*convect([xveloc,yveloc],dt,w)/dt/beta) 
    + on(bb,cc,u=f) // *exp(-r*t); 
);

int ww=1;
for ( n=0; n*dt <= 1.0; n++)
{
    cout <<" iteration 	" << n << " 	j= " << j << endl;
w=u;
eq1;
v = max(u-f,0.);
plot(v,wait=ww);
u = max(u,f);
7.6.1 Stokes and Navier-Stokes

The Stokes equations are: for a given \( f \in L^2(\Omega) \),
\[
\begin{align*}
-\Delta u + \nabla p &= f \\
\nabla \cdot u &= 0
\end{align*}
\]  
\text{in } \Omega \tag{7.42}

where \( u = (u_1, u_2) \) is the velocity vector and \( p \) the pressure. For simplicity, let us choose Dirichlet boundary conditions on the velocity, \( u = u^r \) on \( \Gamma \). In Teman[Theorem 2.2], there ia a weak form of (7.42): Find \( v = (v_1, v_2) \in V(\Omega) \)
\[
V(\Omega) = \{ w \in H^1_0(\Omega)^2 | \text{div} w = 0 \}
\]
which satisfy
\[
\sum_{i=1}^2 \int_{\Omega} \nabla u_i \cdot \nabla v_i = \int_{\Omega} f \cdot w \quad \text{for all } v \in V
\]

Here it is used the existence \( p \in H^1(\Omega) \) such that \( u = \nabla p \), if
\[
\int_{\Omega} u \cdot v = 0 \quad \text{for all } v \in V
\]

Another weak form is derived as follows: We put
\[
V = H^1_0(\Omega)^2; \quad W = \left\{ q \in L^2(\Omega) \left| \int_{\Omega} q = 0 \right. \right\}
\]
By multiplying the first equation in (7.42) with \( v \in V \) and the second with \( q \in W \), subsequent integration over \( \Omega \), and an application of Green’s formula, we have

\[
\int_{\Omega} \nabla u \cdot \nabla v - \int_{\Omega} \text{div} v p = \int_{\Omega} f \cdot v \\
\int_{\Omega} \text{div} u q = 0
\]

This yields the weak form of (7.42): Find \((u, p) \in V \times W\) such that

\[
a(u, v) + b(v, p) = (f, v) \quad (7.43) \\
b(u, q) = 0 \quad (7.44)
\]

for all \((v, q) \in V \times W\), where

\[
a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v = \sum_{i=1}^{2} \int_{\Omega} \nabla u_i \cdot \nabla v_i \quad (7.45) \\
b(u, q) = -\int_{\Omega} \text{div} u q \quad (7.46)
\]

Now, we consider finite element spaces \(V_h \subset V\) and \(W_h \subset W\), and we assume the following basis functions

\[
V_h = V_h \times V_h, \quad V_h = \{v_h | v_h = v_1 \varphi_1 + \cdots + v_M \varphi_M\}, \\
W_h = \{q_h | q_h = q_1 \varphi_1 + \cdots + q_M \varphi_M\}
\]

The discrete weak form is: Find \((u_h, p_h) \in V_h \times W_h\) such that

\[
a(u_h, v_h) + b(v_h, p_h) = (f, v_h), \quad \forall v_h \in V_h \\
b(u_h, q_h) = 0, \quad \forall q_h \in W_h \quad (7.47)
\]

**Note 23** Assume that:

1. There is a constant \(\alpha_h > 0\) such that

\[
a(v_h, v_h) \geq \alpha \|v_h\|_{1,\Omega}^2 \quad \text{for all} \ v_h \in Z_h
\]

where

\[
Z_h = \{v_h \in V_h | b(w_h, q_h) = 0 \quad \text{for all} \ q_h \in W_h\}
\]

2. There is a constant \(\beta_h > 0\) such that

\[
\sup_{v_h \in V_h} \frac{b(v_h, q_h)}{\|v_h\|_{1,\Omega}} \geq \beta_h \|q_h\|_{0,\Omega} \quad \text{for all} \ q_h \in W_h
\]

Then we have an unique solution \((u_h, p_h)\) of (7.47) satisfying

\[
\|u - u_h\|_{1,\Omega} + \|p - p_h\|_{0,\Omega} \leq C \left( \inf_{v_h \in V_h} \|u - v_h\|_{1,\Omega} + \inf_{q_h \in W_h} \|p - q_h\|_{0,\Omega} \right)
\]

with a constant \(C > 0\) (see e.g. [15, Theorem 10.4]).
Let us denote that
\[ A = (A_{ij}), \quad A_{ij} = \int_{\Omega} \nabla \phi_j \cdot \nabla \phi_i \quad i, j = 1, \cdots, M_V \] (7.48)

\[ B = (B_{xi,j}, B_{yi,j}), \quad B_{xi,j} = -\int_{\Omega} \partial \phi_j / \partial x \varphi_i, \quad B_{yi,j} = -\int_{\Omega} \partial \phi_j / \partial y \varphi_i \]

\[ i = 1, \cdots, M_W; j = 1, \cdots, M_V \]

then (7.47) is written by
\[
\begin{pmatrix}
A & B^* \\
B & 0
\end{pmatrix}
\begin{pmatrix}
U_h \\
\{p_h\}
\end{pmatrix}
= 
\begin{pmatrix}
F_h \\
0
\end{pmatrix}
\] (7.49)

where
\[
A = \begin{pmatrix} A & 0 \\ 0 & A \end{pmatrix}, \quad B^* = \begin{pmatrix} B_{xT} \\ B_{yT} \end{pmatrix}, \quad U_h = \begin{pmatrix} \{u_{1,h}\} \\ \{u_{2,h}\} \end{pmatrix}, \quad F_h = \begin{pmatrix} \{\int_{\Omega} f_1 \phi_i\} \\ \{\int_{\Omega} f_2 \phi_i\} \end{pmatrix}
\]

**Penalty method:** This method consists of replacing (7.47) by a more regular problem: Find \((v_{\epsilon h}, p_{\epsilon h}) \in V_h \times \tilde{W}_h\) satisfying
\[
a(u_{\epsilon h}^f, v_h) + b(v_h, p_{\epsilon h}) = (f, v_h), \quad \forall v_h \in V_h
\]
\[
b(u_{\epsilon h}^f, q_h) - \epsilon(p_{\epsilon h}, q_h) = 0, \quad \forall q_h \in \tilde{W}_h
\] (7.50)

where \(\tilde{W}_h \subset L^2(\Omega)\). Formally, we have
\[
\text{div} u_{\epsilon h}^f = \epsilon p_{\epsilon h}^f
\]

and the corresponding algebraic problem
\[
\begin{pmatrix}
A & B^* \\
B & \epsilon I
\end{pmatrix}
\begin{pmatrix}
U_h^f \\
\{p_h^f\}
\end{pmatrix}
= 
\begin{pmatrix}
F_h^f \\
0
\end{pmatrix}
\] (7.51)

**Note 24** We can eliminate \(p_{\epsilon h}^f = (1/\epsilon)B U_h^f\) to obtain
\[
(A + (1/\epsilon)B^*B)U_h^f = F_h^f
\]

Since the matrix \(A + (1/\epsilon)B^*B\) is symmetric, positive-definite, and sparse, (7.51) can be solved by known technique. There is a constant \(C > 0\) independent of \(\epsilon\) such that
\[
\|u_h - u_h^f\|_{1,\Omega} + \|p_h - p_h^f\|_{0,\Omega} \leq C\epsilon
\]

(see e.g. [15, 17.2])

**Example 43 (Cavity.edp)** The driven cavity flow problem is solved first at zero Reynolds number (Stokes flow) and then at Reynolds 100. The velocity pressure formulation is used first and then the calculation is repeated with the stream function vorticity formulation.

We solve the driven cavity problem by the penalty method (7.50) where \(u_{\Gamma} \cdot n = 0\) and \(u_{\Gamma} \cdot s = 1\) on the top boundary and zero elsewhere (\(n\) is the unit normal to \(\Gamma\), and \(s\) the
unit tangent to $\Gamma$.

The mesh is constructed by

```plaintext
mesh Th=square(8,8);
```

We use a classical Taylor-Hood element technic to solve the problem:

The velocity is approximated with the $P_2$ FE ($X_h$ space), and the pressure is approximated with the $P_1$ FE ($M_h$ space),

where

$$X_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \quad v|_K \in P_2 \}$$

and

$$M_h = \{ v \in H^1([0,1]^2) \mid \forall K \in T_h \quad v|_K \in P_1 \}$$

The FE spaces and functions are constructed by

```plaintext
fespace Xh(Th,P2); // definition of the velocity component space
fespace Mh(Th,P1); // definition of the pressure space
Xh u2,v2;
Xh u1,v1;
Xh p,q;
```

The Stokes operator is implemented as a system-solve for the velocity $(u_1,u_2)$ and the pressure $p$. The test function for the velocity is $(v_1,v_2)$ and $q$ for the pressure, so the variational form (7.47) in freefem language is:

```plaintext
solve Stokes (u1,u2,p,v1,v2,q,solver=Crout) =
  int2d(Th) ( dx(u1)*dx(v1) + dy(u1)*dy(v1)
  + dx(u2)*dx(v2) + dy(u2)*dy(v2) )
  - p*q*(0.000001)
  - p*dx(v1) - p*dy(v2)
  - dx(u1)*q - dy(u2)*q
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0); // see Section 3.1.1 for labels 1,2,3,4
```

Each unknown has its own boundary conditions.

If the streamlines are required, they can be computed by finding $\psi$ such that $\text{rot}\psi = u$ or better,

$$-\Delta \psi = \nabla \times u$$

Xh psi,phi;

```plaintext
solve streamlines(psi,phi) =
  int2d(Th) ( dx(psi)*dx(phi) + dy(psi)*dy(phi) )
  + int2d(Th) ( -psi*(dy(u1)-dx(u2)) )
  + on(1,2,3,4,psi=0);
```
Now the Navier-Stokes equations are solved

\[
\frac{\partial u}{\partial t} + u \cdot \nabla u - \nu \Delta u + \nabla p = 0, \quad \nabla \cdot u = 0
\]

with the same boundary conditions and with initial conditions \( u = 0 \).

This is implemented by using the convection operator \texttt{convect} for the term \( \frac{\partial u}{\partial t} + u \cdot \nabla u \), giving a discretization in time

\[
\frac{1}{\tau} (u^{n+1} - u^n \circ X^n) - \nu \Delta u^{n+1} + \nabla p^{n+1} = 0, \quad \nabla \cdot u^{n+1} = 0
\]

(7.52)

The term \( u^n \circ X^n(x) \approx u^n(x - u^n(x)\tau) \) will be computed by the operator “\texttt{convect}”, so we obtain

```c
int i=0;
real nu=1./100.;
real dt=0.1;
real alpha=1/dt;

Xh up1,up2;

problem NS (u1,u2,p,v1,v2,q,solver=Crout,init=i) =
  int2d(Th)(
    alpha*( u1*v1 + u2*v2 )
    + nu * ( dx(u1)*dx(v1) + dy(u1)*dy(v1) 
    + dx(u2)*dx(v2) + dy(u2)*dy(v2) )
    - p*q*(0.000001)
    - p*dx(v1) - p*dy(v2)
    - dx(u1)*q - dy(u2)*q
  )
  + int2d(Th) (-alpha*
    convect([up1,up2],-dt,up1)*v1 -alpha*convect([up1,up2],-dt,up2)*v2 )
  + on(3,u1=1,u2=0)
  + on(1,2,4,u1=0,u2=0)
;

for (i=0;i<=10;i++)
{
  up1=u1;
  up2=u2;
  NS;
  if ( !(i % 10) ) // plot every 10 iteration
    plot(coef=0.2,cmm=" [u1,u2] and p ",p,[u1,u2]);
} 
```

Notice that the stiffness matrices are reused (keyword \texttt{init}=i)
7.6.2 Uzawa Conjugate Gradient

We solve Stokes problem without penalty. The classical iterative method of Uzawa is described by the algorithm (see e.g. [15, 17.3]):

Initialize: Let \( p_0^h \) be an arbitrary chosen element of \( L^2(\Omega) \).

Calculate \( u_h \): Once \( p_n^h \) is known, \( v_n^h \) is the solution of
\[
    u_n^h = A^{-1}(f_h - B^*p_n^h)
\]

Advance \( p_h \): Let \( p_n^{n+1} \) be defined by
\[
p_n^{n+1} = p_n^h + \rho_n B u_n^h
\]

There is a constant \( \alpha > 0 \) such that \( \alpha \leq \rho_n \leq 2 \) for each \( n \), then \( u_n^h \) converges to the solution \( u_h \), and then \( Bv_n^h \to 0 \) as \( n \to \infty \) from the Advance \( p_h \). This method in general converges quite slowly.

First we define mesh, and the Taylor-Hood approximation. So \( X_h \) is the velocity space, and \( M_h \) is the pressure space.

Example 44 (StokesUzawa.edp)

```plaintext
mesh Th=square(10,10);
fespace Xh(Th,P2),Mh(Th,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;  // ppp is a working pressure

varf bx(u1,q) = int2d(Th)( -(dx(u1)*q) );
varf by(u1,q) = int2d(Th)( -(dy(u1)*q) );
varf a(u1,u2) = int2d(Th)( dx(u1)*dx(u2) + dy(u1)*dy(u2) )
    + on(3,u1=1) + on(1,2,4,u1=0) ;
    // remark: put the on(3,u1=1) before on(1,2,4,u1=0)
    // because we want zero on intersection

matrix A= a(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);
matrix By= by(Xh,Mh);

Xh bc1; bc1[] = a(0,Xh);  // boundary condition contribution on u1
Xh bc2; bc2 = 0;  // no boundary condition contribution on u2
Xh b;

p_n^h \to B A^{-1}(-B^*p_n^h) = -\text{div} u_h \text{ is realized as the function divup.}

func real[int] divup(real[int] & pp)
{
    b[] = Bx'*pp; b[] *= -1; b[] += bc1[] ; u1[] = A^-1*b[];  // compute u1(pp)
    b[] = By'*pp; b[] *= -1; b[] += bc2[] ; u2[] = A^-1*b[];  // compute u2(pp)
    u[] = A^-1(Bx'*pp, By'*pp)' T
    ppp[] = Bx*u1[];  // ppp = Bxu1
}
```
ppp[] += By*u2[];  //   +Byu_2
return ppp[] ;
}

Call now the conjugate gradient algorithm:

p=0;q=0;
LinearCG(divup,p[],eps=1.e-6,nbiter=50);  //   \quad p_h^n = 0
// if n > 50 or \left| p_{n+1}^h - p_n^h \right| \leq 10^{-6}, then the loop end.
divup(p[]);  //   compute the final solution
plot([[u1,u2],p,wait=1,value=true,coef=0.1));

7.6.3 NSUzawaCahouetChabart.edp

In this example we solve the Navier-Stokes equation, in the driven-cavity, with the Uzawa algorithm preconditioned by the Cahouet-Chabart method.

The idea of the preconditioner is that in a periodic domain, all differential operators commute and the Uzawa algorithm comes to solving the linear operator \( \nabla ((\alpha Id + \nu \Delta)^{-1} \nabla) \), where \( Id \) is the identity operator. So the preconditioner suggested is \( \alpha \Delta^{-1} + \nu Id \).

To implement this, we reuse the previous example, by including a file. Then we define the time step \( \Delta t \), viscosity, and new variational form and matrix.

Example 45 (NSUzawaCahouetChabart.edp)

```cpp
#include "StokesUzawa.edp"  //   include the Stokes part
real dt=0.05, alpha=1/dt;  //   \Delta t

cout << " alpha = " << alpha;
real xnu=1./400;  //   viscosity \( \nu = \text{Reynolds number}^{-1} \)

//   the new variational form with mass term
varf at(u1,u2)= int2d(Th) ( xnu*dx(u1)*dx(u2)
 + xnu*dy(u1)*dy(u2) + u1*u2*alpha )
 + on(1,2,4,u1=0) + on(3,u1=1) ;

A = at(Xh,Xh,solver=CG);  //   change the matrix

//   set the 2 convect variational form
varf vfconv1(uu,vv) = int2d(Th,qforder=5) (convect([u1,u2],[-dt,u1])*vv*alpha);
varf vfconv2(v2,v1) = int2d(Th,qforder=5) (convect([u1,u2],[-dt,u2])*v1*alpha);

int idt;  //   index of of time set
real temps=0;  //   current time

Mh pprec,prhs;
varf vfMass(p,q) = int2d(Th) (p*q);
matrix MassMh=vfMass(Mh,Mh,solver=CG);

varf vfLap(p,q) = int2d(Th) (dx(pprec)*dx(q)+dy(pprec)*dy(q) + pprec*q*1e-10);
matrix LapMh= vfLap(Mh,Mh,solver=Cholesky);
```
The function to define the preconditioner

```cpp
func real[int] CahouetChabart(real[int] & xx) {
    // xx = f(divu)w_i
    // αLapMh^{-1} + νMassMh^{-1}
    pprec[] = LapMh^{-1} * xx;
    prhs[] = MassMh^{-1} * xx;
    pprec[] = alpha*pprec[] + xnu* prhs[];
    return pprec[];
}
```

The loop in time. Warning with the stop test of the conjugate gradient, because we start from the previous solution and the end the previous solution is close to the final solution, don't take a relative stop test to the first residual, take an absolute stop test (negative here)

```cpp
for (idt = 1; idt < 50; idt++)
{
    temps += dt;
    cout << "--------- temps " << temps << " \n ";
    b1[] = vfconv1(0, Xh);
    b2[] = vfconv2(0, Xh);
    cout << " min b1 b2 " << b1[].min << " " << b2[].min << endl;
    cout << " max b1 b2 " << b1[].max << " " << b2[].max << endl;
    // call Conjuged Gradient with preconditioner ’
    // warning eps < 0 => absolute stop test
    LinearCG(divup, p[], eps=-1.e-6, nbiter=50, precon=CahouetChabart);
    divup(p[]);
    // computed the velocity
    plot([u1, u2], p, wait=!((idt%10), value=1, coef=0.1));
}
```

### 7.7 Domain decomposition

We present, three classical examples, of domain decomposition technique: first, Schwarz algorithm with overlapping, second Schwarz algorithm without overlapping (also call Shur complement), and last we show to use the conjugate gradient to solve the boundary problem of the Shur complement.

#### 7.7.1 Schwarz Overlap Scheme

To solve

\[-\Delta u = f, \quad \text{in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_{\Gamma} = 0\]

the Schwarz algorithm runs like this

\[-\Delta u_1^{n+1} = f \quad \text{in } \Omega_1 \quad u_1^{n+1}|_{\Gamma_1} = u_2^n\]
\[-\Delta u_2^{n+1} = f \quad \text{in } \Omega_2 \quad u_2^{n+1}|_{\Gamma_2} = u_1^n\]

where $\Gamma_i$ is the boundary of $\Omega_i$ and on the condition that $\Omega_1 \cap \Omega_2 \neq \emptyset$ and that $u_i$ are zero at iteration 1.
Here we take $\Omega_1$ to be a quadrangle, $\Omega_2$ a disk and we apply the algorithm starting from zero.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig7.19.png}
\caption{The 2 overlapping mesh $TH$ and $th$}
\end{figure}

\textbf{Example 46 (Schwarz-overlap.edp)}

\begin{verbatim}
int inside = 2;  // inside boundary
int outside = 1;  // outside boundary
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=inside;};
border e1(t=pi/2, 2 * pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
shape th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
shape TH = buildmesh( e(5*n) + e1(25*n) );
plot(th,TH,wait=1);  // to see the 2 meshes

The space and problem definition is :

\begin{verbatim}
fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH)  ( dx(U)*dx(V)+dy(U)*dy(V) )
    + int2d(TH) ( -V ) + on(inside,U = u) + on(outside,U= 0 ) ;
problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th)  ( dx(u)*dx(v)+dy(u)*dy(v) )
    + int2d(th) ( -v ) + on(inside ,u = U) + on(outside,u = 0 ) ;
\end{verbatim}

The calculation loop:

\begin{verbatim}
for ( i=0 ;i< 10; i++) {
\end{verbatim}
\end{verbatim}
7.7.2 Schwarz non Overlap Scheme

To solve
\[-\Delta u = f \text{ in } \Omega = \Omega_1 \cup \Omega_2 \quad u|_\Gamma = 0,\]
the Schwarz algorithm for domain decomposition without overlapping runs like this

Let introduce $\Gamma_i$ is common the boundary of $\Omega_1$ and $\Omega_2$ and $\Gamma_i^c = \partial\Omega_i \setminus \Gamma_i$.
The problem find $\lambda$ such that $(u_1|_{\Gamma_i} = u_2|_{\Gamma_i})$ where $u_i$ is solution of the following Laplace problem:
\[-\Delta u_i = f \text{ in } \Omega_i \quad u_i|_{\Gamma_i} = \lambda \quad u_i|_{\Gamma_i^c} = 0\]
To solve this problem we just make a loop with upgrading \( \lambda \) with

\[
\lambda = \lambda \pm \frac{(u_1 - u_2)}{2}
\]

where the sign + or − of \( \pm \) is choose to have convergence.

**Example 47 (Schwarz-no-overlap.edp)**

```plaintext
// schwarz1 without overlapping

int inside = 2;
int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, 1){ x = 1-t; y = t;label=inside;};
border e1(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;

mesh th = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
mesh TH = buildmesh ( e(5*n) + e1(25*n) );
plot(th,TH,wait=1,ps="schwarz-no-u.eps");

fespace vh(th,P1);
fespace VH(TH,P1);
vh u=0,v; VH U,V;
vh lambda=0;

int i=0;

problem PB(U,V,init=i,solver=Cholesky) =
    int2d(TH) ( dx(U)*dx(V)+dy(U)*dy(V) )
    + int2d(TH) ( -V)
    + int1d(TH,inside)(-lambda*V) + on(outside,U= 0 ) ;

problem pb(u,v,init=i,solver=Cholesky) =
    int2d(th) ( dx(u)*dx(v)+dy(u)*dy(v) )
    + int2d(th) ( -v)
    + int1d(th,inside)(+lambda*v) + on(outside,u = 0 ) ;

for ( i=0 ;i< 10; i++)
{
    PB;
    pb;
    lambda = lambda - (u-U)/2;
    plot(U,u,wait=true);
}

plot(U,u,ps="schwarz-no-u.eps");
```
7.7.3 Schwarz-gc.edp

To solve

\[-\Delta u = f \quad \text{in} \quad \Omega = \Omega_1 \cup \Omega_2, \quad u|_{\Gamma} = 0,\]

the Schwarz algorithm for domain decomposition without overlapping runs like this. Let introduce \(\Gamma_i\) is common the boundary of \(\Omega_1\) and \(\Omega_2\) and \(\Gamma_i^c = \partial \Omega_i \setminus \Gamma_i\). The problem find \(\lambda\) such that \((u_1|_{\Gamma_i} = u_2|_{\Gamma_i})\) where \(u_i\) is solution of the following Laplace problem:

\[-\Delta u_i = f \quad \text{in} \quad \Omega_i, \quad u_i|_{\Gamma_i^c} = 0, \quad u_i|_{\Gamma_i} = \lambda\]

The version of this example for Shur component. The border problem is solved with conjugate gradient.

First, we construct the two domain

**Example 48 (Schwarz-gc.edp)**

```cpp
// Schwarz without overlapping (Shur complement Neumann -> Dirichlet)
real cpu=clock();
int inside = 2;
int outside = 1;

border Gamma1(t=1,2){x=t;y=0;label=outside;};
border Gamma2(t=0,1){x=2;y=t;label=outside;};
border Gamma3(t=2,0){x=t;y=1;label=outside;};
border GammaInside(t=1,0){x = 1-t; y = t;label=inside;};
border GammaArc(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;

// build the mesh of \(\Omega_1\) and \(\Omega_2\)
mesh Th1 = buildmesh( Gamma1(5*n) + Gamma2(5*n) + GammaInside(5*n) + Gamma3(5*n) );
mesh Th2 = buildmesh( GammaInside(-5*n) + GammaArc(25*n) );
plot(Th1,Th2);

// defined the 2 FE space
```
7.7. DOMAIN DECOMPOSITION

fespace Vh1(Th1,P1), Vh2(Th2,P1);

Note 25 It is impossible to define a function just on a part of boundary, so the lambda function must be defined on the all domain $\Omega_1$ such as

$$v_{h1}(\thetah_1,p_1), v_{h2}(\thetah_2,p_1);$$

$\text{Note 25}$ It is impossible to define a function just on a part of boundary, so the lambda function must be defined on the all domain $\Omega_1$ such as

$$Vh1 \text{ lambda=0;} \quad // \quad \text{take } \lambda \in V_{h1}$$

The two Poisson problem:

$$Vh1 u_1,v_1; \quad Vh2 u_2,v_2; \quad \text{int } i=0; \quad // \quad \text{for factorization optimization}$$

$$\text{problem } Pb1(u_1,v_1,\text{init}=i,\text{solver}=\text{Cholesky}) =$$

$$\quad \text{int2d(Th1)( } dx(u_1) \ast dx(v_1)+dy(u_1) \ast dy(v_1) \text{ )}$$
$$\quad + \text{int2d(Th1)( } -v_1 \text{ )}$$
$$\quad + \text{int1d(Th1,inside)( } +\lambda \ast v_1 \text{ )} + \text{on(outside,u_1=0 )};$$

$$\text{problem } Pb2(u_2,v_2,\text{init}=i,\text{solver}=\text{Cholesky}) =$$

$$\quad \text{int2d(Th2)( } dx(u_2) \ast dx(v_2)+dy(u_2) \ast dy(v_2) \text{ )}$$
$$\quad + \text{int2d(Th2)( } -v_2 \text{ )}$$
$$\quad + \text{int1d(Th2,inside)( } -\lambda \ast v_2 \text{ )} + \text{on(outside,u_2=0 )};$$

or, we define a border matrix, because the lambda function is none zero inside the domain $\Omega_1$:

$$\varf b(u_2,v_2,\text{solver}=\text{CG}) =\text{int1d(Th1,inside)( } u_2 \ast v_2 \text{ )};$$

$$\text{matrix } B= b(Vh1,Vh1,\text{solver}=\text{CG});$$

The boundary problem function,

$$\lambda \rightarrow \int_{\Gamma_i} (u_1 - u_2) v_1$$

$$\text{func real[int] BoundaryProblem(real[int] \&l)}$$

$$\{$$
$$\quad \lambda[] = l; \quad // \quad \text{make FE function form } l$$
$$\quad Pb1; \quad Pb2;$$
$$\quad i++; \quad // \quad \text{no refactorization } i !=0$$
$$\quad v1=-(u1-u2);$$
$$\quad \lambda[] = B \ast v1[];$$
$$\quad \text{return } \lambda[];$$

$$\}$$

$$\text{Note 26}$$ The difference between the two notations $v_1$ and $v1[]$ is: $v_1$ is the finite element function and $v1[]$ is the vector in the canonical basis of the finite element function $v1$.

$$Vh1 p=0,q=0; \quad // \quad \text{solve the problem with Conjugue Gradient}$$

$$\text{LinearCG(BoundaryProblem,p[],eps=1.e-6,nbiter=100); \quad // \quad \text{compute the final solution, because CG works with increment}}$$

$$\text{BoundaryProblem(p[]);} \quad // \quad \text{solve again to have right u1,u2}$$

cout << " -- CPU time schwarz-gc:" << clock()-cpu << endl;

$$\text{plot(u1,u2); \quad // \quad plot}$$
7.8 Fluid/Structures Coupled Problem

This problem involves the Lamé system of elasticity and the Stokes system for viscous fluids with velocity $u$ and pressure $p$:

$$-\Delta u + \nabla p = 0, \ \nabla \cdot u = 0, \ \text{in} \ \Omega, \ u = u_\Gamma \ \text{on} \ \Gamma = \partial \Omega$$

where $u_\Gamma$ is the velocity of the boundaries. The force that the fluid applies to the boundaries is the normal stress $h = (\nabla u + \nabla u^T) n - pn$.

Elastic solids subject to forces deform: a point in the solid, originally at $(x,y)$ goes to $(X,Y)$ after. When the displacement vector $v = (v_1,v_2) = (X-x,Y-y)$ is small, Hooke’s law relates the stress tensor $\sigma$ inside the solid to the deformation tensor $\epsilon$:

$$\sigma_{ij} = \lambda \delta_{ij} \nabla \cdot v + 2\mu \epsilon_{ij}, \ \epsilon_{ij} = \frac{1}{2}(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_i})$$

where $\delta$ is the Kronecker symbol and where $\lambda, \mu$ are two constants describing the material mechanical properties in terms of the modulus of elasticity, and Young’s modulus.

The equations of elasticity are naturally written in variational form for the displacement vector $v(x) \in V$ as

$$\int_\Omega [2\mu \epsilon_{ij}(v)\epsilon_{ij}(w) + \lambda \epsilon_{ii}(v)\epsilon_{jj}(w)] = \int_\Omega g \cdot w + \int_\Gamma h \cdot w, \forall w \in V$$

The data are the gravity force $g$ and the boundary stress $h$.

Example 49 (Fluidstruct.edp) In our example the Lamé system and the Stokes system are coupled by a common boundary on which the fluid stress creates a displacement of the boundary and hence changes the shape of the domain where the Stokes problem is integrated. The geometry is that of a vertical driven cavity with an elastic lid. The lid is a beam with weight so it will be deformed by its own weight and by the normal stress due to the fluid reaction. The cavity is the $10 \times 10$ square and the lid is a rectangle of height $l = 2$.

A beam sits on a box full of fluid rotating because the left vertical side has velocity one. The beam is bent by its own weight, but the pressure of the fluid modifies the bending. The bending displacement of the beam is given by $(uu,vv)$ whose solution is given as follows.

```plaintext
// Fluid-structure interaction for a weighting beam sitting on a // square cavity filled with a fluid.
int bottombeam = 2;  // label of bottombeam
border a(t=2,0) { x=0; y=t ;label=1;};  // left beam
border b(t=0,10) { x=t; y=0 ;label=bottombeam;};  // bottom of beam
border c(t=0,2) { x=10; y=t ;label=1;};  // right beam
border d(t=0,10) { x=10-t; y=2; label=3;};  // top beam
real E = 21.5;
real sigma = 0.29;
real mu = E/(2*(1+sigma));
```
real lambda = E*sigma/((1+sigma)*(1-2*sigma));
real gravity = -0.05;
mesh th = buildmesh( b(20)+c(5)+d(20)+a(5));
fespace Vh(th,P1);
Vh uu,w,vv,s, fluidforce=0;
cout << "lambda,mu,gravity ="<<lambda<< " " << mu << " " << gravity << endl;

solve bb((uu,vv),[w,s]) =
  int2d(th)(
    2*mu*(dx(uu)*dx(w)+ ((dx(vv)+dy(uu))*(dx(s)+dy(w)))/4 )
    + lambda*(dx(uu)+dy(vv))*(dx(w)+dy(s))/2
  )
  + int2d(th) (-gravity*s)
  + on(1,uu=0,vv=0)
  + fluidforce[];

plot((uu,vv),wait=1);

Then Stokes equation for fluids ast low speed are solved in the box below the beam, but the beam has deformed the box (see border h):

// Stokes on square b,e,f,g driven cavite on left side g
border e(t=0,10) { x=t; y=-10; label= 1; };
// bottom
border f(t=0,10) { x=10; y=-10+t; label= 1; };
// right
border g(t=0,10) { x=0; y=-t; label= 2; };
// left
border h(t=0,10) { x=t; y=vv(t,0)*( t>=0.001)*(t <= 9.999); // top of cavity deforme
  label=3;};

mesh sh = buildmesh(h(-20)+f(10)+e(10)+g(10));
plot(sh,wait=1);

We use the Uzawa conjugate gradient to solve the Stokes problem like in example Section 7.6.2

fespace Xh(sh,P2),Mh(sh,P1);
Xh u1,u2,v1,v2;
Mh p,q,ppp;

varf bx(u1,q) = int2d(sh)( -(dx(u1)*q));
varf by(u1,q) = int2d(sh)( -(dy(u1)*q));

varf Lap(u1,u2)= int2d(sh)( dx(u1)*dx(u2) + dy(u1)*dy(u2) )
  + on(2,u1=1) + on(1,3,u1=0) ;

Xh bc1; bc1[] = Lap(0,Xh);
Xh brhs;

matrix A= Lap(Xh,Xh,solver=CG);
matrix Bx= bx(Xh,Mh);
matrix By= by(Xh,Mh);
Xh bcx=0,bcy=1;
func real[int] divup(real[int] & pp)
{
    int verb=verbosity;
    verbosity=0;
    brhs[] = Bx'*pp; brhs[] += bc1[] .*bcx[];
    u1[] = A^-1*brhs[];
    brhs[] = By'*pp; brhs[] += bc1[] .*bcy[];
    u2[] = A^-1*brhs[];
    ppp[] = Bx*u1[];
    ppp[] += By*u2[];
    verbosity=verb;
    return ppp[];
}

p=0;q=0;u1=0;v1=0;
LinearCG(divup,p[],eps=1.e-3,nbiter=50);
divup(p[]);

Now the beam will feel the stress constraint from the fluid:

Vh sigma11,sigma22,sigma12;
Vh uu1=uu,vv1=vv;
sigma11([x+uu,y+vv]) = (2 * dx(u1)-p);
sigma22([x+uu,y+vv]) = (2 * dy(u2)-p);
sigma12([x+uu,y+vv]) = (dx(u1)+dy(u2));

which comes as a boundary condition to the PDE of the beam:

varf fluidf([uu,vv],[w,s]) fluidforce =
solve  bbst([uu,vv],[w,s],init=i) =
    int2d(th) (  
        2*mu*(dx(uu)*dx(w)+ ((dx(vv)+dy(uu)) * (dx(s)+dy(w)))/4 )
        + lambda*(dx(uu)+dy(vv)) * (dx(w)+dy(s))/2
    )
    + int2d(th) (-gravity*s)
    + int1d(th,bottombeam) ( -coef*( sigma11*N.x*w + sigma22*N.y*s
        + sigma12*(N.y*w+N.x*s) ) )
    + on (1,uu=0,vv=0);
plot([uu,vv],wait=1);
real err = sqrt(int2d(th) ( (uu-uu1)^2 + (vv-vv1)^2 ));
cout << " Erreur L2 = " << err << "--------\n";

Notice that the matrix generated by bbst is reused (see init=i). Finally we deform the beam

th1 = movemesh(th, [x+0.2*uu, y+0.2*vv]);
plot(th1,wait=1);
7.9 Transmission Problem

Consider an elastic plate whose displacement changes vertically, which is made up of three plates of different materials, welded on each other. Let \( \Omega_i \), \( i = 1, 2, 3 \) be the domain occupied by the \( i \)-th material with tension \( \mu_i \) (see Section 7.1.1). The computational domain \( \Omega \) is the interior of \( \bigcup_{i=1}^{3} \Omega_i \). The vertical displacement \( u(x,y) \) is obtained from

\[
- \mu_i \Delta u = f \quad \text{in } \Omega_i \quad \text{ (7.53)}
\]

\[
\mu_i \partial_n u|_{\Gamma_i} = -\mu_j \partial_n u|_{\Gamma_j} \quad \text{on } \Omega_i \cap \Omega_j \quad \text{if } 1 \leq i < j \leq 3 \quad \text{ (7.54)}
\]

where \( \partial_n u|_{\Gamma_i} \) denotes the value of the normal derivative \( \partial_n u \) on the boundary \( \Gamma_i \) of the domain \( \Omega_i \).

By introducing the characteristic function \( \chi_i \) of \( \Omega_i \), that is,

\[
\chi_i(x) = 1 \quad \text{if } x \in \Omega_i; \quad \chi_i(x) = 0 \quad \text{if } x \notin \Omega_i \quad \text{(7.55)}
\]

we can easily rewrite (7.53) and (7.54) to the weak form. Here we assume that \( u = 0 \) on \( \Gamma = \partial \Omega \).

The problem Transmission: For a given function \( f \), find \( u \) such that

\[
a(u,v) = \ell(f,v) \quad \text{for all } v \in H_0^1(\Omega) \quad \text{(7.56)}
\]

\[
a(u,v) = \int_{\Omega} \mu \nabla u \cdot \nabla v, \quad \ell(f,v) = \int_{\Omega} f v
\]

where \( \mu = \mu_1 \chi_1 + \mu_2 \chi_2 + \mu_3 \chi_3 \). Here we notice that \( \mu \) becomes the discontinuous function.

With dissipation, and at the thermal equilibrium, the temperature equation is:

This example explains the definition and manipulation of region, i.e. subdomains of the whole domain.

Consider this L-shaped domain with 3 diagonals as internal boundaries, defining 4 subdomains:

```plaintext
// example using region keyword
// construct a mesh with 4 regions (sub-domains)

border a(t=0,1) {x=t; y=0;};
border b(t=0,0.5) {x=1; y=t;};
border c(t=0,0.5) {x=1-t; y=0.5;};
border d(t=0.5,1) {x=0.5; y=t;};
border e(t=0.5,1) {x=1-t; y=1;};
border f(t=0,1) {x=0; y=1-t;};

border i1(t=0,0.5) {x=t; y=1-t;};
border i2(t=0,0.5) {x=t; y=t;};
border i3(t=0,0.5) {x=1-t; y=t;};

mesh th = buildmesh (a(6) + b(4) + c(4) + d(4) + e(4) + f(6) + i1(6) + i2(6) + i3(6));

fespace Ph(th,P0); // constant discontinuous functions / element
fespace Vh(th,P1); // P_1 continuous functions / element

Ph reg=region; // defined the P_0 function associated to region number
plot(reg,fill=1,wait=1,value=1);
```
\textbf{region} is a keyword of freefem++ which is in fact a variable depending of the current position (is not a function today, use \texttt{Ph reg=region;} to set a function). This variable value returned is the number of the subdomain of the current position. This number is defined by "buildmesh" which scans while building the mesh all its connected component. So to get the number of a region containing a particular point one does:

\begin{verbatim}
int nupper=reg(0.4,0.9);  // get the region number of point (0.4,0.9)
int nlower=reg(0.9,0.1);  // get the region number of point (0.4,0.1)
cout << " nlower " << nlower << " , nupper = " << nupper<< endl;

Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
\end{verbatim}

This is particularly useful to define discontinuous functions such as might occur when one part of the domain is copper and the other one is iron, for example. We this in mind we proceed to solve a Laplace equation with discontinuous coefficients ($\nu$ is 1, 6 and 11 below).

\begin{verbatim}
Ph nu=1+5*(region==nlower) + 10*(region==nupper);
plot(nu,fill=1,wait=1);
problem lap(u,v) = int2d(th) ( nu*( dx(u)*dx(v)*dy(u)*dy(v) ) ) + int2d(-1*v) + on(a,b,c,d,e,f,u=0);
plot(u);
\end{verbatim}
7.10 Free Boundary Problem

The domain $\Omega$ is defined with:

```plaintext
real L=10; // longueur du domaine
real h=2.1; // hauteur du bord gauche
real h1=0.35; // hauteur du bord droite

border a(t=0,L){x=t;y=0;}; // bottom: $\Gamma_a$
border b(t=0,h1){x=L;y=t;};// right: $\Gamma_b$
border f(t=L,0){x=t;y=t*(h1-h)/L+h}; // free surface: $\Gamma_f$
border d(t=h,0){x=0;y=t}; // left: $\Gamma_d$
```

int n=4;
mesh Th=buildmesh (a(10*n)+b(6*n)+f(8*n)+d(3*n));
plot(Th,ps="dTh.eps");

The free boundary problem is:

Find $u$ and $\Omega$ such that:
\[
\begin{cases}
-\Delta u = 0 & \text{in } \Omega \\
u = y & \text{on } \Gamma_b \\
\frac{\partial u}{\partial n} = 0 & \text{on } \Gamma_d \cup \Gamma_a \\
\frac{\partial u}{\partial n} = \frac{q}{K} n_x & \text{and } u = y & \text{on } \Gamma_f
\end{cases}
\]

We use a fixed point method; \( \Omega^0 = \Omega \)
in two step, first we solve the classical following problem:

\[
\begin{cases}
-\Delta u = 0 & \text{in } \Omega^n \\
u = y & \text{on } \Gamma^n_b \\
\frac{\partial u}{\partial n} = 0 & \text{on } \Gamma^n_d \cup \Gamma^n_a \\
u = y & \text{on } \Gamma^n_f
\end{cases}
\]

The variational formulation is:

find \( u \) on \( V = H^1(\Omega^n) \), such than \( u = y \) on \( \Gamma^n_b \) and \( \Gamma^n_f \)

\[
\int_{\Omega^n} \nabla u \nabla u' = 0, \quad \forall u' \in V \text{ with } u' = 0 \text{ on } \Gamma^n_b \cup \Gamma^n_f
\]

and secondly to construct a domain deformation \( \mathcal{F}(x, y) = [x, y - v(x, y)] \)

where \( v \) is solution of the following problem:

\[
\begin{cases}
-\Delta v = 0 & \text{in } \Omega^n \\
v = 0 & \text{on } \Gamma^n_a \\
\frac{\partial v}{\partial n} = 0 & \text{on } \Gamma^n_b \cup \Gamma^n_d \\
\frac{\partial v}{\partial n} = \frac{\partial u}{\partial n} - \frac{q}{K} n_x & \text{on } \Gamma^n_f
\end{cases}
\]

The variational formulation is:

find \( v \) on \( V \), such than \( v = 0 \) on \( \Gamma^n_a \)

\[
\int_{\Omega^n} \nabla v \nabla v' = \int_{\Gamma^n_f} \left( \frac{\partial u}{\partial n} - \frac{q}{K} n_x \right) v', \quad \forall v' \in V \text{ with } v' = 0 \text{ on } \Gamma^n_a
\]

finally the new domain \( \Omega^{n+1} = \mathcal{F}(\Omega^n) \)

**Example 50 (freeboundary.edp)** The *FreeFem++* implementation is:

```plaintext
real q=0.02;    // flux entrant
real K=0.5;     // permeabilité

defspace Vh(Th,P1);
defint j=0;

Vh u,v,uu,vv;

problem Pu(u,uu,solver=CG) = int2d(Th) (dx(u)*dx(uu)+dy(u)*dy(uu))
+ on(b,f,u=y);
```
7.10. FREE BOUNDARY PROBLEM

The problem is defined as follows:

\[ Pv(v,vv, solver=CG) = \text{int2d}(Th) \{ dx(v) \times dx(vv) + dy(v) \times dy(vv) \} \]
\[ \text{on} \ (a, v=0) + \text{int1d}(Th,f) (vv*(q/K)*N.y - (dx(u)*N.x+dy(u)*N.y)) \];

Initial conditions

```plaintext
real errv=1;
real erradap=0.001;
verbosity=1;
while (errv>1e-6)
{
    j++;
    Pu;
    Pv;
    plot(Th,u,v,wait=0);
    errv=int1d(Th,f)(v*v);
    real coef=1;
}
```

Mesh adaptation and remeshing

```plaintext
real mintcc = checkmovemesh(Th,[x,y])/5.;
real mint = checkmovemesh(Th,[x,y-v*coef]);
if (mint<mintcc || j%10==0) {
    Th=adaptmesh(Th,u, err=erradap);
    mintcc = checkmovemesh(Th,[x,y])/5.;
}
```

Mesh adaptation continues

```plaintext
while (1)
{
    real mint = checkmovemesh(Th,[x,y-v*coef]);
    if (mint<mintcc) break;
}
```

Calculating deformation

```plaintext
Th=movemesh(Th,[x,y-coef*v]);
cout << "\n\n"<<j <<"---------------- errv = " << errv << "\n\n";
```

Plotting results

```plaintext
plot(Th,ps="d_Thf.eps");
plot(u,wait=1,ps="d_u.eps");
```

---

**Figure 7.27:** The final solution on the new domain \( \Omega \)
7.11 Nolinear-elas.edp

The nonlinear elasticity problem is find the displacement \((u_1, u_2)\) minimizing \(J\)

\[
\min J(u_1, u_2) = \int_{\Omega} f(F2) - \int_{\Gamma_p} P_a u_2
\]

where \(F2(u_1, u_2) = A(E[u_1, u_2], E[u_1, u_2])\) and \(A(X, Y)\) is bilinear sym. positive form with respect two matrix \(X, Y\). where \(f\) is a given \(C^2\) function, and \(E[u_1, u_2] = (E_{ij})_{i=1,2, j=1,2}\) is the Green-Saint Venant deformation tensor defined with:

\[
E_{ij} = 0.5(\partial_i u_j + \partial_j u_i) + \sum_k \partial_i u_k \times \partial_j u_k
\]

The differential of \(J\) is

\[
DJ(u_1, u_2)(v_1, v_2) = \int 2A(E[u_1, u_2], DE[u_1, u_2](v_1, v_2))f'(F2(u_1, u_2))) - \int_{\Gamma_p} P_a u_2
\]

denote \(u = u_1, u_2, v = v_1, v_2, w = (w_1, w_2)\) and the second order differential is

\[
D^2 J(u)((v), (w)) = A(E[u], DE[u](v))A(E[u], DE[u](w))f''(F2(u))) + A(DE[u](v), DE[u](w))f'(F2(u))) + A(DE[u], D^2 E[u]((v), (w)))f'(F2(u)))
\]

where \(DE\) and \(D^2 E\) are the first and second differential of \(E\).

The Newton Method is

choose \(n = 0\), and \(u_0, v_0\) the initial displacement

- loop:
  - find \((du, dv)\) : solution of
    \[
    D^2 J(u_n, v_n)((w, s), (du, dv)) = DJ(u_n, v_n)(w, s), \quad \forall w, s
    \]
  - \(un = un - du, \quad vn = vn - dv\)
  - until \((du, dv)\) small is enough
The way to implement this algorithm in freefem++ is use a macro tool to implement $A$ and $F_2$, $f$, $f'$, $f''$.

A macro is like is ccpp preprocessor of C++, but this begin by macro and the end of the macro definition is the begin of the comment //. In this case the macro is very useful because the type of parameter can be change. And it is easy to make automatic differentiation.

```cpp
// non linear elasticity model
// -------------------------------
// with huge utilisation of macro
// optimize version
// -------------------------------
// problem is find $(u,u,v)$ minimizing $J$
// $minJ(u,u,v) = int f(F_2) - int Pa * uu$
// $dJ(u,u,v,v) = int dF_2(u,v,uu,vv) df(F_2(u,v))$
// where $F_2 = (\frac{1}{2} EAE)$ ,
// $E_0(U) = \frac{1}{2}(\nabla U + \nabla U^t + \nabla U^t \nabla U)$
// $(u_1)$
// with $U = ( )$
// $(u_2)$
// so:

(1) \[ E_{ij} = 0.5(d_iu_j + d_ju_i) + \sum_k d_iu_k * d_j * u_k \]

// the 3 components of the Green Saint Venant deformation tensor:
// $E_1(u_1,u_2) = E_{11}$
// $E_2(u_1,u_2) = E_{12} = E_{21}$
// $E_3(u_1,u_2) = E_{22}$

// remark : we can parametrize $E_1,E_2,E_3$ with:
// $EE(da,db,a,b,u_1,u_2)$
// where $da,db$ correspond to $d_i,d_j$ in (1)
// where $a,b$ correspond to $u_i,u_j$ in (1)
// where $u_1,u_2$ correspond to $u_1,u_2$ in (1)
// ----------------------------------------------
// first the linear part of $EE$ linear elasticite
// remark a macro end with a // comment
macro EEL(di,dj,ui,uj) ( (di(uj)+dj(ui)) * 0.5 ) // 11

// non linear par of $EE$ (bilinear) simple to differential
macro bEEENL(di,dj,u1,u2,v1,v2) (di(u1)*dj(v1)*.5+di(u2)*dj(v2)*.5)

macro EENL(di,dj,u1,u2) bEEENL(di,dj,u1,u2,u1,u2)
macro dEEENL(di,dj,u1,u2,du1,du2) ( bEEENL(di,dj,du1,du2,u1,u2) + bEEENL(di,dj,u1,u2,du1,du2) )

macro EE(di,dj,ui,uj,u1,u2) (EEL(di,dj,ui,uj) + EEL(di,dj,ui,u2))
macro dEE(di,dj,ui,duj,u1,u2,du1,du2) (EEL(di,dj,duj,du1,u2) + EEL(di,dj,ui,du1,du2))
macro ddEE(di,dj,du1,du2,ddu1,ddu2) (dEE(di,dj,du1,du2,ddu1,ddu2))
```
remark:
dEE(d_{i,j}, d_{u,v}, d_{u,j}, d_{u,l}, u_1, u_2, d_u, d_v) is "the formal differential of EE"
where dulu = $\delta u_1$, dudu = $\delta^2 u_2$

ddEE(d_{i,j}, d_{u,v}, d_{u,j}, d_{u,l}, u_1, u_2, d_u, d_v) is "the formal differential of dEE"
where dddu = $\delta^2 u_1$, dduu = $\delta^2 u_2$

---

the macro corresponding to the 3 components of E

macro E1(u,v) /*E1*/ (dx, dx, u, u, v) //
macro E2(u,v) /*E2*/ (dx, dy, u, v, u, v) //
macro E3(u,v) /*E3*/ (dy, dy, v, v, u, v) //

macro dE1(u,v,uu,vv) /*dE1*/ (dx, dx, uu, uu, u, v, uu, vv, vv) //
macro dE2(u,v,uu,vv) /*dE2*/ (dx, dy, uu, uu, vv, v, uu, vv) //
macro dE3(u,v,uu,vv) /*dE3*/ (dy, dy, uu, uu, vv, vv, v, vv) //

macro ddE1(u,v,uu,vv,uuu,vvv) /*ddE1*/ (dx, dx, uu, vv, uuu, vvv) //
macro ddE2(u,v,uu,vv,uuu,vvv) /*ddE2*/ (dx, dy, uu, vv, uuu, vvv) //
macro ddE3(u,v,uu,vv,uuu,vvv) /*ddE3*/ (dy, dy, uu, vv, uuu, vvv) //

---

// a formal bilinear term

macro PP(A,B,u,v) (A(u,v) * B(u,v)) //

// a formal diff bilinear term

macro dPP(A,B,dA,dB,u,v,uu,vv) (dA(u,v,uu,vv) * B(u,v) + A(u,v) * dB(u,v,uu,vv)) //

// a formal diff² bilinear term

macro ddPP(A,B,dA,dB,dD,u,v,uu,vv,uuu,vvv) \\
  (dA(u,v,uu,vv) * dB(u,v,uuu,vvv) + dB(u,v,uuu,vvv) * dA(u,v,uu,vv) \\
   + ddA(u,v,uu,vv,uuu,vvv) * B(u,v) + A(u,v) * ddB(u,v,uu,vv,uuu,vvv)) //

// so the matrix A is 6 coef

//

// a11 a12 a13
// a12 a22 a23
// a13 a23 a33

macro F2(u,v) /* F2 */ (a11*PP(E1,E1,u,v) + a12*PP(E1,E2,u,v) + a13*PP(E1,E3,u,v) + a21*PP(E2,E1,u,v) + a22*PP(E2,E2,u,v) + a23*PP(E2,E3,u,v) + a31*PP(E3,E1,u,v) + a32*PP(E3,E2,u,v) + a33*PP(E3,E3,u,v)) // end macro F2

macro dF2(u,v,uu,vv) /* dF2 */ (a11*dPP(E1,E1,dE1,dE1,u,v,uu,vv) + a12*dPP(E1,E2,dE1,dE2,u,v,uu,vv) + a13*dPP(E1,E3,dE1,dE3,u,v,uu,vv) + a21*dPP(E2,E1,dE2,dE1,u,v,uu,vv) + a22*dPP(E2,E2,dE2,dE2,u,v,uu,vv) + a23*dPP(E2,E3,dE2,dE3,u,v,uu,vv) + a31*dPP(E3,E1,dE3,dE1,u,v,uu,vv) + a32*dPP(E3,E2,dE3,dE2,u,v,uu,vv) + a33*dPP(E3,E3,dE3,dE3,u,v,uu,vv))
+ a32*dPP(E3,E2,dE3,dE2,u,v,uu,vv)
+ a33*dPP(E3,E3,dE3,dE3,u,v,uu,vv)
) // end macro dF2 (DF2)

macro ddf2(u,v,uu,vv,uuu,vvv) /* ddf2 */ (  
   a11=ddPP(E1,E1,dE1,dE1,ddE1,ddE1,u,v,uu,vv,uuu,vvv)
+ a12=ddPP(E1,E2,dE1,dE2,ddE1,ddE2,u,v,uu,vv,uuu,vvv)
+ a13=ddPP(E1,E3,dE1,dE3,ddE1,ddE3,u,v,uu,vv,uuu,vvv)
+ a21=ddPP(E2,E1,dE2,dE1,ddE2,ddE1,u,v,uu,vv,uuu,vvv)
+ a22=ddPP(E2,E2,dE2,dE2,ddE2,ddE2,u,v,uu,vv,uuu,vvv)
+ a23=ddPP(E2,E3,dE2,dE3,ddE2,ddE3,u,v,uu,vv,uuu,vvv)
+ a31=ddPP(E3,E1,dE3,dE1,ddE3,ddE1,u,v,uu,vv,uuu,vvv)
+ a32=ddPP(E3,E2,dE3,dE2,ddE3,ddE2,u,v,uu,vv,uuu,vvv)
+ a33=ddPP(E3,E3,dE3,dE3,ddE3,ddE3,u,v,uu,vv,uuu,vvv)
) // end macro ddf2 (D^2F2)

// differential of J:
// for hyper elasticity problem
// ------------------------------

macro f(u) (u) // end of macro
macro df(u) (1) // end of macro df = f'
macro ddf(u) (0) // end of macro ddf = f''

// -- du caouchouc --- CF cours de Herve Le Dret.
// -------------------------------
real mu = 0.012e5; // kg/cm^2
real lambda = 0.4e5; // kg/cm^2

//
// σ = 2µE + λtr(E)Id
//
// ( a b )
// ( b c )
//
// tr*Id : (a,b,c) -> (a+c,0,a+c)
// so the associated matrix is:
// ( 1 0 1 )
// ( 0 0 0 )
// ( 1 0 1 )
//
// ---------------------- the coef
real a11= 2*mu + lambda ;
real a22= 2*mu ;
real a33= 2*mu + lambda ;
real a12= 0 ;
real a13= lambda ;
real a23= 0 ;
// symetric part
real a21= a12 ;
real a31= a13 ;
real a32= a23 ;
real Pa=1e2; // a pressure of 100 Pa

int n=30,m=10;
mesh Th= square(n,m,[x,.3*y]); // label: 1 bottom, 2 right, 3 up, 4 left;
```
int bottom=1, right=2, upper=3, left=4;

plot(Th);

fespace Wh(Th,P1dc);

fespace Vh(Th,[P1,P1]);

fespace Sh(Th,P1);

Wh e2,fe2, dfe2, ddfe2;   // optimisation
Wh ett, ezz, err, erz;     // optimisation

Vh [uu,vv], [w,s],[un,vn];
[un,vn]=[0,0];            // intialisation
[uu,vv]=[0,0];

varf vmass([[uu,vv],[w,s]],solver=CG) = int2d(Th)( uu*w + vv*s );
matrix M=vmass(Vh,Vh);

problem NonLin([[uu,vv],[w,s]],solver=LU)=
  int2d(Th,qforder=1) (   // (D^2J(un)) part
    ddf2(un,vn,uu,vv,w,s)* dfe2
    + dF2(un,vn,uu,vv)* dF2(un,vn,w,s)* ddfe2
  )
  - int2d(Th,<1) (       // (DJ(un)) part
    dF2(un,vn,w,s)* dfe2
  )
  - int1d(Th,3)(Ps*sn)
  + on(right,left,uu=0,vv=0);

// Newton’s method
// ---------------

Sh u1,v1;

for (int i=0;i<10;i++)
{
  cout << "Loop " << i << endl;
  e2 = F2(un,vn);
  dfe2 = df(e2);
  ddfe2 = ddf(e2);
  cout << " e2 max " << e2[].max << " min " << e2[].min << endl;
  cout << " dfe2 max " << dfe2[].max << " min " << dfe2[].min << endl;
  cout << " ddfe2 max " << ddfe2[].max << " min " << ddfe2[].min << endl;
  NonLin;  // compute [uu,vv] = (D^2J(un))^{-1}(DJ(un))

  w[] = M*uu[];
  real res = sqrt(w[].’*uu[]);  // norme L^2of[uu,vv]
  ul = uu;
  v1 = vv;
  cout << " L^2 residual = " << res << endl;
  cout << " ul min = " << ul[].min << " ul max = " << ul[].max << endl;
  cout << " v1 min = " << v1[].min << " v1 max = " << v1[].max << endl;

  plot([uu,vv],wait=1,cmm=" uu, vv ");
  un[] = uu[];
  plot([un,vn],wait=1,cmm=" deplacement ");
  if (res<1e-5) break;
}

plot([un,vn],wait=1);
```
7.11. NOLINEAR-ELAS.EDP

```
mesh th1 = movemesh(Th, [x+un, y+vn]);
plot(th1, wait=1); // see figure 7.29
```

Figure 7.29: The deformated domain
Chapter 8

Parallel version experimental

A first test of parallisation of FreeFem++ is make under mpi. We add three word in the language:

- **mpisize** The total number of processes
- **mpirank** the number of my current process in \(\{0, \ldots, \text{mpisize} - 1\}\).
- **processor** a function to set the possessor to send or receive data
- **broadcast** a function to broadcast from a processor to all other a data

```cpp
processor(10) << a; // send to the process 10 the data a;
processor(10) >> a; // receive from the process 10 the data a;
```

8.1 Schwarz in parallel

If example is just the rewritting of example schwarz-overlap in section 7.7.1. How to use

```bash
[examples++-mpi] hecht% lamboot
```

LAM 6.5.9/MPI 2 C++/ROMIO - Indiana University

```bash
[examples++-mpi] hecht% mpirun -np 2 FreeFem++-mpi schwarz-c.edp
```

```cpp
// a new coding version c, methode de schwarz in parallele
// with 2 proc.
// -------------------------------
// F.Hecht december 2003
// -------------------------------
// to test the broadcast instruction
// and array of mesh
// add add the stop test
// -------------------------------
```
if ( mpisize != 2 ) {
    cout << " sorry number of processeur !=2 " << endl;
    exit(1);
}

verbosity=3;
real pi=4*atan(1);
int inside = 2;
int outside = 1;
border a(t=1,2){x=t;y=0;label=outside;};
border b(t=0,1){x=2;y=t;label=outside;};
border c(t=2,0){x=t ;y=1;label=outside;};
border d(t=1,0){x = 1-t; y = t;label=inside;};
border e(t=0, pi/2){ x= cos(t); y = sin(t);label=inside;};
border el(t=pi/2, 2*pi){ x= cos(t); y = sin(t);label=outside;};
int n=4;
mesh[int] Th(mpisize);
if (mpirank == 0)
    Th[0] = buildmesh( a(5*n) + b(5*n) + c(10*n) + d(5*n));
else
    Th[1] = buildmesh ( e(5*n) + el(25*n) );

broadcast(processor(0),Th[0]);
broadcast(processor(1),Th[1]);

fespace Vh(Th[mpirank],P1);
fespace Vhother(Th[1-mpirank],P1);

Vh u=0,v;
Vhother U=0;
int i=0;

problem pb(u,v,init=i,solver=Cholesky) =
    int2d(Th[mpirank])( dx(u)*dx(v)+dy(u)*dy(v) )
    - int2d(Th[mpirank])( v)
    + on(inside,u = U ) + on(outside,u= U ) ;

for ( i=0 ;i< 20; i++)
{
    cout << mpirank << " loop " << i << " << endl;
    pb;
    // send u to the other proc, receive in U
    processor(1-mpirank) << u[]; processor(1-mpirank) >> U[];
    real err0,err1;
    err0 = int1d(Th[mpirank],inside)(square(U-u)) ;
    // send err0 to the other proc, receive in err1
    processor(1-mpirank)<<err0; processor(1-mpirank)>>err1;
    cout <<" err = " " err " " err0 = " " err0 " " , err1 = " " err1 << endl;
    if(err<1e-3) break;
}
if (mpirank==0)
    plot(u,U,ps="uU.eps");
Chapter 9

Graphical User Interface

There are two different graphical user interfaces available for freem++:

- **FreeFem++-cs** is part of the standard freem++ package. It runs on Linux, MacOS X and Windows.
- **FFedit** is available as a separate package. It runs on Linux and MacOS X and requires tcl/tk to be installed.

### 9.1 FreeFem++-cs

“-cs” stands for “client/server”. The executable program named FreeFem++-cs is the client. It automatically starts a server program named FreeFem++-cs-server every time the user asks for a script to be run. To run FreeFem++-cs, just type its name in, optionally followed by a script name. It will open a window corresponding to fig. 9.1.

The main characteristics of FreeFem++-cs are:

- A main window composed of three panels: editor with syntax highlighting (right), freem++ messages (bottom) and graphics (left).
- Panel sizes can be changed. Any of the three panels can be made to fill the whole window.
- The edited script can be run at any time by clicking on the “Run” button.
- Graphics can be examined (e.g. zoomed) while freem++ is running.
- A running freem++ computation can be paused or stopped at any time.

All commands should be self-explanatory. Here are just a few useful hints:

- There is no need to save a freem++ script to run it. It is run exactly as displayed in the editor window, and the corresponding file is not touched.
CHAPTER 9. GRAPHICAL USER INTERFACE

Figure 9.1: FreeFem++-cs main window

- The current directory is updated every time a script is loaded or saved. All include directives are therefore relative to the directory where the main script is located.

- Specifying `wait=1` in a `plot` command is exactly equivalent to clicking on the “Pause” button when the plot is displayed.

- In zoom mode, if your mouse has more than one button, a middle-click resets any zooming coefficient, and a right-click zooms in the opposite way of the left-click.

9.2 FFedit

FFedit runs on Linux and MacOSX.

9.2.1 Installation of Tcl and Tk

You have to install tcl8.4.6 and tk8.4.6 for the GUI to work. First go to http://www.tcl.tk/software/tcltk/downloadnow84.tml and download:
tcl8.4.6-src.tar.gz and tk8.4.6-src.tar.gz
Then do:
tar zxvf tcl8.4.6-src.tar.gz
9.2. **FFEDIT**

```
tar zxvf tk8.4.6-src.tar.gz
It creates two directories : tcl8.4.6 and tk8.4.6
Now do :
cd tcl8.4.6
cd unix (if your OS is Linux or MacOSX)
./configure
make
make install

then

cd tk8.4.6

cd unix

./configure
make
make install
```

At the end of installation, you have to find where is your binary “wish” or “wish84” or “wish8.4” by typing :

```
-> which wish (or wish84 or wish8.4)
```

If wish84 does exist it is all. If not, you have to go in the directory where is “wish” or “wish8.4” (for example /usr/bin/)
and then create a link :

```
-> ln -s wish wish84
```
or

```
-> ln -s wish8.4 wish84
```

### 9.2.2 Description

The Graphic User Interface is in the directory called FFedit. You can run it by typing ./FFedit.tcl

The Graphic User Interface shows a text window with buttons on the left and right side, a horizontal menu bar above and an entry below where you can see the path of the script when it is opened or where you can type the path of a script to run it.

This is the description of the different functions of the GUI :

*New : you can access it by the button “New” on the right side or by selecting it in the menu File on the horizontal bar. It deletes the current script and enables you to type a new script.

*Open : you can access it by the button “Open” on the right side or by selecting it in the menu File or by typing simultaneously ctrl+o on the keyboard. It enables you to select a script which has been saved. This script is then opened in the text window. Then, you can modify it, save the changes, save under another name or run it.

*Save : you can access it by the button “Save” on the right side or by selecting it in the
menu File or by typing simultaneously ctrl+s on the keyboard. It enables you to save the changes of a script.

*Save as : you can access it by the button “Save As” on the right side or by selecting it in the menu File.
It enables you to locate where you want to save your script and to choose your the name of your script.

*Run : you can access it by the button “Run” on the right side or by selecting it in the menu File or by typing simultaneously ctrl+r on the keyboard.
It enables you to run the current script.
Warning: when you run by typing ctrl+r, the cursor must be in the text window.

*Print : you can access it by the button “Print” on the right side or by selecting it in the menu File or by typing simultaneously ctrl+p on the keyboard.
It enables you to print your script. You have to choose the name of the printer.

*Help (under construction) : you can access it by the button “Help” on the left side. When an example is opened, it shows you a documentation about this example.
Warning : The bottom of each page is not accessible, you have to print to see the entire document.

*Example : you can access it by the button “Ex” on the left side. It runs a little example.

*Read Mesh : you can access it by the button “R.M” on the left side. It enables you to read a mesh which has been saved. It adds the command in your script so that you can use it in your script.

*Polygonal Border : you can access it by the button “P.B” on the left side.
It enables you to build a polygonal border.
When you click on this button, a new window is opened : you have to click on the button “Border” then it asks you how many borders you want. When you enter a number and click on “OK” the exact number of couple of entries enable you to enter the coordinates of the vertices. And then you have to enter the number of points on each border. The border number 1 is the segment between the vertex number 1 and the vertex number 2 ...etc...
You must turn in the opposite sens of needles of a watch.
When you have finished, you have to click on the button “Build”. It builds the domain with polygonal border and shows the result.
Then you can save the result by clicking on the button “S.Mesh”. Choose the extension .msh for the name.

*Navier Stokes : you can access it by the button “N.S” on the left side.
A new window is opened. You have to build your polygonal domain by clicking on the button “Border” it works like the Polygonal Border function.
You can save by clicking on the button “S.Mesh”. Choose the extension .msh for the name.
Then you have to choose the Limits Condition by clicking on the buton “L.C”.
First choose between “Free” or “Imposed’, then click on the button “Validate” and enter
the expression of Imposed condition. You have to enter the tangential and the normal component of the velocity. Then click on “Validate” again. The expression of the limit condition can be a number but a mathematical expression as well.

*emc2 : you can access it by the button “emc2”. It runs emc2 the mesh building software (http://www-rocq1.inria.fr/gamma/cdrom/www/emc2/fra.htm)

*Set up : you can access it by the button “set up”. It is the first thing you have to do when you use FFedit for the first time. When you click on this button, a new window is opened, with two entries where you have to enter the path of the binary of FreeFem++ (where you compiled or installed) and the path of the scripts (programs written with FreeFem++) for instance the examples provided in FreeFem++.

*Undo : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+z. It is an unlimited undo function.

*Redo : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+e. It is an unlimited redo function.

*Cut : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+x on the keyboard.

*Copy : you can access it by selecting in the menu Edit or by typing simultaneously ctrl+c on the keyboard.

*Paste : you can access it by selectiog in the menu Edit or by typing simultaneously ctrl+y on the keyboard.

*Delete : you can access it by selecting in the Edit menu. It is a Delete function.

*Select all : you can access it by selecting in the Edit menu or by typing simultaneously ctrl+l. This function select all the text you have written, so you can delete, cut copy paste etc...

*Background : you can access it by selecting in the menu Color. You can then choose the color of the background.

*Foreground : you can access it by selecting in the menu Color. You can then choose the color of the foreground.

*Syntax Color : you can access it by selecting in the menu Color. It enables the syntax coloring of your FreeFem++ script.

*Family : you can access it by selecting in the menu Font. You can then choose the style of
your characters.

*Size : you can access it by selecting in the menu Font. You can then choose the size of your characters.

*Find : you can access it by selecting in the menu Search. It enables you to find a word in the whole text. (This function doesn’t work yet.)

*Find next : you can access it by selecting in the menu Search. It enables you to find a word from the position of the cursor. (This function doesn’t work yet.)

*Replace : you can access it by selecting in the menu Search. It enables you to replace a word by another in the whole text.

### 9.2.3 Cygwin version

This version is for Windows.

### 9.2.4 Installation of Cygwin

First you have to go to the site:
http://www.cygwin.com
Click on “Install or Update now”
A window appears.
Click on “Open” then “Next” and then choose “Install from Internet”
Click on “Next” twice and choose “Direct Connection”.
Then choose one of the download sites.
For instance : ftp://ftp-stud.fht-esslingen.de
Then choose in each category the options to install.
(click on the symbol + for each, you can then see the options appear)

Here are the required options for FreeFem++ and TCL TK to work :
+ all options of “Devel” (it includes gcc ...etc...)
+ all options of “Graphics” (specially “Gnuplot” to be able to see the results of FreeFem++ on a graphic)
+ all options of X11 and XFree86

You can install “Xemacs” and others editors in the category “Editors”.

Then click on “Next” and wait that Cygwin is installed.
Then an icône appears on yours Desktop.
Click on it and the Cygwin window is opened. It is like an Unix terminal.

To work on a terminal X type :
- > startX
9.2. **FFEDIT**

You will have to work on a terminal X to have Gnuplot and visualize the results of FreeFem++ scripts.

Now you have to compile and install FreeFem++.

### 9.2.5 Compilation and Installation of FreeFem++ under Cygwin

go to the site:  
http://www.freefem.org  
click on FreeFem++  
and download FreeFem++ (the version when I wrote this is 1.38)  
Choose to download in your cygwin/home/you directory.

Then on your terminal Cygwin type:
- `tar zxvf freefem++.tgz`
  
to uncompress this file.

Go into FreeFem++v1.38 directory by typing:
- `cd FreeFem++v1.38`

Now you have to compile FreeFem++.

type:  
- `make all HOSTTYPE=i-386`
  
to compile FreeFem++

If there is an error: “... -ldl : no such file or directory”
Then you have to modify the Makefile-i386 which is in the directory src:
- `cd src`
Edit it (with xemacs for example):
- `xemacs Makefile-i386`
  
at line 1 : replace “LIBLOCAL = -ldl” by “#LIBLOCAL = -ldl”
  
It will comment this line because -ldl is not on your machine.
Then return to the main directory
- `cd`
and type:
- `make all HOSTTYPE=i-386`

At the end of compilation, a directory called “c-i386” is created.
In this directory you can find the binary FreeFem++.

You can now run an example:
First open an X terminal:
- `startX`
In this terminal go in to FreeFem++v1.38:
- `cd FreeFem++v1.38`
and type:
9.2.6 Compilation and Installation of tcl8.4.0 and tk8.4.0 under Cygwin

Now if you want to use the Graphical User Interface of FreeFem++ (called FFedit)
you have to install the language TCL TK in which FFedit has been written.
The version which works under cygwin is tcl8.4.0 and tk8.4.0
The latest version when I wrote this is tcl8.4.6 and tk8.4.6

DON’T USE IT

It works under Linux and MacOsX but not under Cygwin.

You have to download tcl8.4.0 and tk8.4.0 :
For instance, go to Google.fr and type download tcl tk 8.4.0
And choose the “Sourceforge.net: Project Filelist”.
Choose :
tcl8.4.0-src.tar.gz and tk8.4.0-src.tar.gz
When the download is finished you have to uncompress these directories:
- > tar zxvf tcl8.4.0-src.tar.gz
- > tar zxvf tk8.4.0-src.tar.gz

tcl8.4.0 and tk8.4.0 will work under Cygwin only if you apply
a patch on both:
These patches are on the site:
http://www.xraylith.wisc.edu/ khan/software/tcl
Choose “Tcl/Tk8.4.0 for Cygwin
Click on “very preliminary Cygwin ports of Tcl/Tk8.4.0
You are then on the site ftp
Follow the instructions of the README or follow these instructions:

1) Run :
- > xemacs tcl-8.4.0-cygwin.diff
By doing this, you create a new file called “tcl-8.4.0-cygwin.diff”
on the site ftp click on “tcl-8.4.0-cygwin.diff”
Do a Copy/Paste of the contain into your xemacs window and save it.

2) Do the same with “tk-8.4.0-cygwin.diff”

Now you have to apply the patch in tcl8.4.0 and tk8.4.0
The two previous patch files (.diff) must be respectively
in tcl8.4.0 and tk8.4.0 directories.
9.2. **FFEDIT**

```bash
-> cp tcl-8.4.0-cygwin.diff tcl8.4.0
cp tcl-8.4.0-cygwin.diff tk8.4.0
(If the two files are one level under tcl8.4.0 and tk8.4.0)

Now apply the patches:
type:

-> cd tcl8.4.0
cd
-> patch -p0 -s < tcl-8.4.0-cygwin.diff
cd
cd tk8.4.0
-> patch -p0 -s < tk-8.4.0-cygwin.diff

Now you can compile and install TCL TK under Cygwin:

* compilation and installation of tcl8.4.0

Go in to the directory tcl8.4.0/win
-> cd tcl8.4.0
cd
-> cd win
Then type:
-> ./configure
The two steps remaining are make and make install
type
-> make
The compilation starts, when finished install by typing:
-> make install

When finished try to see if it works by typing:
-> tclsh84
if ok quit by typing ctrl-c

*Compilation and installation of tk8.4.0

Go in to the directory tk8.4.0/win
-> cd tk8.4.0
cd
-> cd win
Then type:
-> ./configure
The two steps remaining are make and make install
type
-> make
The compilation starts, if you have errors like:

windres -o tk.res.o –include “C:/cygwin/home/ly/tk8.4.0/generic”
–include “C Option-I is deprecated for setting the input format, please use -J instead”
windres : can’t open icon file ’tk.ico’ : no such file or directory

This file ’tk.ico’ is in fact in the directory win/rc
You have to copy it in the directory ’generic’:
Be in tk8.4.0
type :
-> cp win/rc/tk.ico generic/

If you compile again you will see that there is the same
ersors with the files:
“buttons.bmp” “cursor00.cur” “cursor02.cur” ...etc...
“wish.exe.manifest” and “wish.ico”

Do the same for these files.
For the cursor*.cur files do once the command:
-> cp win/rc/cursor*.cur generic/

when finished install by typing:
-> make install

When finished try to see if it works by typing:
-> wish84
if ok quit by typing ctrl-c

9.2.7 Use

Now everything is ok to use FFedit and FreeFem++ under Windows by Cygwin.
WARNING: if you work on the Cygwin terminal you will not be able
to see the graphical results of FreeFem++.

You have to run an X terminal and run FFedit under this X terminal:

To run an X terminal under cygwin type on your Cygwin terminal:
-> startX

An X terminal runs:
Under this terminal:
type
-> cd FFedit
-> ./FFedit.tcl
Chapter 10

Mesh Files

10.1 File mesh data structure

The mesh data structure, output of a mesh generation algorithm, refers to the geometric
data structure and in some case to another mesh data structure.

In this case, the fields are

- **MeshVersionFormatted** 0

- **Dimension** (I) *dim*

- **Vertices** (I) *NbOfVertices*

  \[ \begin{array}{l}
  \left( \left( \mathbb{R} \ x_j^i, \ j=1, \text{dim} \right), \ (I) \ Ref\phi^v_i, \ i=1, \text{NbOfVertices} \right) \\
  \end{array} \]

- **Edges** (I) *NbOfEdges*

  \[ \begin{array}{l}
  \left( \ @@Vertex_1^i, @@Vertex_2^i, \ (I) \ Ref\phi^e_i, \ i=1, \text{NbOfEdges} \right) \\
  \end{array} \]

- **Triangles** (I) *NbOfTriangles*

  \[ \begin{array}{l}
  \left( \ @@Vertex_j^i, \ j=1,3 \right), \ (I) \ Ref\phi^t_i, \ i=1, \text{NbOfTriangles} \right) \\
  \end{array} \]

- **Quadrilaterals** (I) *NbOfQuadrilaterals*

  \[ \begin{array}{l}
  \left( \ @@Vertex_j^i, \ j=1,4 \right), \ (I) \ Ref\phi^t_i, \ i=1, \text{NbOfQuadrilaterals} \right) \\
  \end{array} \]

- **Geometry**

  \begin{array}{l}
  (C*) \ FileNameOfGeometricSupport \\
  \end{array} \]

  - **VertexOnGeometricVertex**

    \[ \begin{array}{l}
    (I) \ NbOfVertexOnGeometricVertex \\
    \left( \ @@Vertex_i, @@Vertex_{geo}^i, \ i=1, \text{NbOfVertexOnGeometricVertex} \right) \\
    \end{array} \]

  - **EdgeOnGeometricEdge**

    \[ \begin{array}{l}
    (I) \ NbOfEdgeOnGeometricEdge \\
    \left( \ @@Edge_i, @@Edge_{geo}^i, \ i=1, \text{NbOfEdgeOnGeometricEdge} \right) \\
    \end{array} \]

- **CrackedEdges** (I) *NbOfCrackedEdges*

  \[ \begin{array}{l}
  \left( \ @@Edge_1^i, @@Edge_2^i, \ i=1, \text{NbOfCrackedEdges} \right) \\
  \end{array} \]
When the current mesh refers to a previous mesh, we have in addition

- **MeshSupportOfVertices**  
  (C*) FileNameOfMeshSupport

  - **VertexOnSupportVertex**  
    (I) NbOfVertexOnSupportVertex  
    ( @@Vertex$i$ , @@Vertex$supp_i$ , $i=1$ , NbOfVertexOnSupportVertex )

- **VertexOnSupportEdge**  
  (I) NbOfVertexOnSupportEdge  
  ( @@Vertex$i$ , @@Edge$supp_i$ , (R) $u_i^{supp}$ , $i=1$ , NbOfVertexOnSupportEdge )

- **VertexOnSupportTriangle**  
  (I) NbOfVertexOnSupportTriangle  
  ( @@Vertex$i$ , @@Tria$supp_i$ , (R) $u_i^{supp}$ , (R) $v_i^{supp}$ , $i=1$ , NbOfVertexOnSupportTriangle )

- **VertexOnSupportQuadrilaterals**  
  (I) NbOfVertexOnSupportQuadrilaterals  
  ( @@Vertex$i$ , @@Quad$supp_i$ , (R) $u_i^{supp}$ , (R) $v_i^{supp}$ , $i=1$ , NbOfVertexOnSupportQuadrilaterals )

### 10.2 bb File type for Store Solutions

The file is formatted such that:

```
2 nbsol nbv 2  
((U$ij$ , $\forall i \in \{1,...,nbsol\}$ , $\forall j \in \{1,...,nbv\}$)
```

where

- **nbsol** is a integer equal to the number of solutions.
- **nbv** is a integer equal to the number of vertex.
- **U$ij$** is a real equal the value of the $i$ solution at vertex $j$ on the associated mesh background if read file, generated if write file.

### 10.3 BB File Type for Store Solutions

The file is formatted such that:

```
2 n typesol$1$ ... typesol$n$ nbv 2  
(((U$ij$ , $\forall i \in \{1,...,typesol^k\}$ , $\forall k \in \{1,...n\}$ , $\forall j \in \{1,...,nbv\}$)
```

where

- **n** is a integer equal to the number of solutions
- **typesol$^k$** , type of the solution number $k$, is
10.4 Metric File

A metric file can be of two types, isotropic or anisotropic. The isotrope file is such that
\[ n_{bv} \quad 1 \quad h_i \quad \forall i \in \{1, \ldots, n_{bv}\} \]
where

- \( n_{bv} \) is a integer equal to the number of vertices.
- \( h_i \) is the wanted mesh size near the vertex \( i \) on background mesh, the metric is \( M_i = h_i^{-2} I_d \), where \( I_d \) is the identity matrix.

The metric anisotrope
\[ n_{bv} \quad 3 \quad a_{11}, a_{21}, a_{22} \quad \forall i \in \{1, \ldots, n_{bv}\} \]
where

- \( n_{bv} \) is a integer equal to the number of vertices,
- \( a_{11}, a_{21}, a_{22} \) is metric \( M_i = \left( \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right) \) which define the wanted mesh size in a vicinity of the vertex \( i \) such that \( h \) in direction \( u \in \mathbb{R}^2 \) is equal to \( |u|/\sqrt{u \cdot M_i u} \), where \( \cdot \) is the dot product in \( \mathbb{R}^2 \), and \( |\cdot| \) is the classical norm.

10.5 List of AM_FMT, AMDBA Meshes

The mesh is only composed of triangles and can be defined with the help of the following two integers and four arrays:
\[ n_{bt} \quad \text{is the number of triangles.} \]
\[ n_{bv} \quad \text{is the number of vertices.} \]
\[ n(1:3, 1:nbt) \quad \text{is an integer array giving the three vertex numbers counterclockwise for each triangle.} \]
\textit{c(1:2,nbv)} is a real array giving the two coordinates of each vertex.

\textit{refs(nbv)} is an integer array giving the reference numbers of the vertices.

\textit{reft(nbv)} is an integer array giving the reference numbers of the triangles.

**AM_FMT Files**  In fortran the \texttt{am_fmt} files are read as follows:

\begin{verbatim}
open(1, file='xxx.am_fmt', form='formatted', status='old')
read (1, *) nbv, nbt
read (1, *) ((nu(i,j), i=1,3), j=1, nbt),
& ((c(i,j), i=1,2), j=1, nbv),
& ( reft(i), i=1, nbt),
& ( refs(i), i=1, nbv)
close(1)
\end{verbatim}

**AM Files**  In fortran the \texttt{am} files are read as follows:

\begin{verbatim}
open(1, file='xxx.am', form='unformatted', status='old')
read (1, *) nbv, nbt
read (1) ((nu(i,j), i=1,3), j=1, nbt),
& ((c(i,j), i=1,2), j=1, nbv),
& ( reft(i), i=1, nbt),
& ( refs(i), i=1, nbv)
close(1)
\end{verbatim}

**AMDBA Files**  In fortran the \texttt{amdba} files are read as follows:

\begin{verbatim}
open(1, file='xxx.amdba', form='formatted', status='old')
read (1, *) nbv, nbt
read (1, *) (k, (c(i,k), i=1,2), refs(k), j=1, nbv)
read (1, *) (k, (nu(i,k), i=1,3), reft(k), j=1, nbt)
close(1)
\end{verbatim}

**msh Files**  First, we add the notions of boundary edges

\texttt{nbbe} is the number of boundary edge.

\texttt{nube(1:2,1:nbbe)} is an integer array giving the two vertex numbers

\texttt{refbe(1:nbbe)} is an integer array giving the two vertex numbers

In fortran the \texttt{msh} files are read as follows:

\begin{verbatim}
open(1, file='xxx.msh', form='formatted', status='old')
read (1, *) nbv, nbt, nbbe
read (1, *) ((c(i,k), i=1,2), refs(k), j=1, nbv)
read (1, *) ((nu(i,k), i=1,3), reft(k), j=1, nbt)
read (1, *) ((ne(i,k), i=1,2), refbe(k), j=1, nbbe)
close(1)
\end{verbatim}
**ftq Files**  In fortran the ftq files are read as follows:

```fortran
open(1, file='xxx.ftq', form='formatted', status='old')
read (1, *) nbv, nbe, nbt, nbq
read (1, *) (k(j), (nu(i, j), i=1, k(j)), ref(t(j), j=1, nbe)
read (1, *) ((c(i, k), i=1, 2), refs(k), j=1, nbv)
close(1)
```

where if \( k(j) = 3 \) then the element \( j \) is a triangle and if \( k = 4 \) the the element \( j \) is a quadrilateral.
Chapter 11

Add new finite element

11.1 Some notation

For a function $f$ taking value in $\mathbb{R}^N$, $N = 1, 2, \cdots$, we define the finite element approximation $\Pi_h f$ of $f$. Let us denote the number of the degrees of freedom of the finite element by $NbDoF$. Then the $i$-th base $\omega^K_i (i = 0, \cdots, NbDoF - 1)$ of the finite element space has the $j$-th component $\omega^K_{ij}$ for $j = 0, \cdots, N - 1$.

The operator $\Pi_h$ is called the interpolator of the finite element. We have the identity $\omega^K_i = \Pi_h \omega^K_i$.

Formally, the interpolator $\Pi_h$ is constructed by the following formula:

$$\Pi_h f = \sum_{k=0}^{kP_i-1} \alpha_k f_{j_k} (P_{p_k}) \omega^K_{i_k}$$

(11.1)

where $P_p$ is a set of $npPi$ points.

In the formula (11.1), the list $p_k, j_k, i_k$ depend just on the type of finite element (not on the element), but the coefficient $\alpha_k$ can be depending on the element.

Example 1: classical scalar Lagrange finite element, first we have $kPi = npPi = NbOfNode$ and

- $P_p$ is the point of the nodal points
- the $\alpha_k = 1$, because we take the value of the function at the point $P_k$
- $p_k = k$, $j_k = k$ because we have one node per function.
- $j_k = 0$ because $N = 1$

Example 2: The Raviart-Thomas finite element:

$$RT0_h = \{ v \in H(div) / \forall K \in T_h \ v|_K(x,y) = |^{\alpha_K}_{\beta_K} + \gamma_K |_y \}$$

(11.2)

The degree of freedom are the flux throw an edge $e$ of the mesh, where the flux of the function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is $\int_e f.n_e$, $n_e$ is the unit normal of edge $e$ (this implies a orientation of all the edges of the mesh, for example we can use the global numbering of the edge vertices and we just go to small to large number).
To compute this flux, we use an quadrature formular with one point, the middle point of the
dge. Consider a triangle \( T \) with three vertices \((a, b, c)\). Let denote the vertices numbers by
\( i_a, i_b, i_c \), and define the three edge vectors \( e_0, e_1, e_2 \) by \( sgn(i_b - i_c)(b - c) \), \( sgn(i_c - i_a)(c - a) \),
\( sgn(i_a - i_b)(a - b) \).
The three basis functions are:
\[
\omega^K_0 = \frac{sgn(i_b - i_c)(x - a)}{2|T|}, \quad \omega^K_1 = \frac{sgn(i_c - i_a)(x - b)}{2|T|}, \quad \omega^K_2 = \frac{sgn(i_a - i_b)(x - c)}{2|T|},
\]
where \( |T| \) is the area of the triangle \( T \).
So we have \( N = 2, kPi = 6; npPi = 3; \) and:

- \( P_p = \{ \frac{b+c}{2}, \frac{a+c}{2}, \frac{b+a}{2} \} \)
- \( \alpha_0 = -e_2^0, \alpha_1 = e_1^0, \alpha_2 = -e_2^1, \alpha_3 = e_1^1, \alpha_4 = -e_2^2, \alpha_5 = e_1^2 \) (effectively, the vector
  \( (-e_2^m, e_1^m) \) is orthogonal to the edge \( e_m = (e_1^m, e_2^m) \) with a length equal to the side of
  the edge or equal to \( \int_{e_m} \).
- \( i_k = \{0, 0, 1, 1, 2, 2\} \),
- \( p_k = \{0, 0, 1, 1, 2, 2\} \), \( j_k = \{0, 1, 0, 1, 0, 1, 0, 1\} \).

### 11.2 Which class of add

Add file FE_ADD.cpp in directory src/femlib for exemple first to initialize :

```cpp
#include "error.hpp"
#include "rgraph.hpp"
using namespace std;
#include "RNM.hpp"
#include "fem.hpp"
#include "FESpace.hpp"

namespace Fem2D {

Second, you are just a class which derive for public TypeOfFE like:

class TypeOfFE_RTortho : public TypeOfFE { public:
    static int Data[]; // some numbers
    TypeOfFE_RTortho();
    TypeOfFE( 0+3+0, // nb degree of freedom on element
                    2, // dimension N of vectorial FE (1 if scalar FE)
            Data, // the array data
                    1, // nb of subdivision for plotting
                    1, // nb of sub finite element (generaly 1)
                    6, // number kPi of coef to build the interpolator [11.1]
            3, // number npPi of integration point to build interpolator
                0 // an array to store the coef \( \alpha_k \) to build interpolator
        // here this array is no constant so we have
        // to rebuilt for each element.
    );

    
};
```
11.2. WHICH CLASS OF ADD

const R2 Pt[] = { R2(0.5,0.5), R2(0.0,0.5), R2(0.5,0.0) };  // the set of Point in \( \hat{K} \)

for (int p=0, kk=0; p<3; p++) {
    P_Pi_h[p]=Pt[p];
    for (int j=0; j<2; j++)
        pij_alpha[kk++]= IPJ(p,p,j);        // definition of \( i_k,p_k,j_k \) in (11.1)
}

void FB(const bool * watdd, const Mesh & Th, const Triangle & K, const R2 &PHat, RNMK__ & val) const;

void Pi_h_alpha(const baseFElement & K, KN_<double> & v) const;

where the array data is form with the concatenation of five array of size NbDoF and one array of size N.
This array is:

```
int TypeOfFE_RTortho::Data[] = {
    // for each df 0,1,3 :
    3,4,5,   // the support of the node of the df
    0,0,0,   // the number of the df on the node
    0,1,2,   // the node of the df
    0,0,0,   // the df come from which FE (generaly 0)
    0,1,2,   // which are de df on sub FE
    0,0 };   // for each component \( j = 0, N-1 \) it give the sub FE associated
```

where the support is a number 0, 1, 2 for vertex support, 3, 4, 5 for edge support, and finally 6 for element support.
The function to defined the function \( \omega^K_i \), this function return the value of all the basics function or this derivatives in array val, computed at point PHat on the reference triangle corresponding to point R2 \( P = K(\text{Phat}) \); on the current triangle K.
The index \( i,j,k \) of the array val\((i,j,k)\) corresponding to:

- \( i \) is basic function number on finite element \( i \in [0, N_{\text{f}}] \)
- \( j \) is the value of component \( j \in [0, N[ \)
- \( k \) is the type of computed value \( f(P), dx(f)(P), dy(f)(P), ... i \in [0, \text{last_operatortype}] \).

Remark for optimisation, this value is computed only if \( \text{whatd}[k] \) is true, and the numbering is defined with

```
enum operatortype { op_id=0,
    op_dx=1, op_dy=2,
    op_dxx=3, op_dyy=4,
    op_dyx=5, op_dxy=5,
    op_dz=6,
    op_dzz=7,
    op_dzx=8, op_dxz=8,
    op_dzy=9, op_dyz=9
};
const int last_operatortype=10;
```
The shape function:

```cpp
void TypeOfFE_RTortho::FB(const bool * whatd, const Mesh & Th, const Triangle & K, const R2 & PHat, RNMK_ & val) const
{
    R2 P(K(PHat));
    R2 A(K[0]), B(K[1]), C(K[2]);
    R l0=1-P.x-P.y, l1=P.x, l2=P.y;
    assert(val.N() >=3);
    assert(val.M()==2);
    val=0;
    R a=1./(2*K.area);
    R a0= K.EdgeOrientation(0) * a ;
    R a1= K.EdgeOrientation(1) * a ;
    R a2= K.EdgeOrientation(2) * a ;

    if (whatd[op_id]) // value of the function
    {
        assert(val.K()>op_id);
        RN_ f0(val('.',0,0));
        RN_ f1(val('.',1,0));
        f1[0] = (P.x-A.x)*a0;
        f0[0] = -(P.y-A.y)*a0;
        f1[1] = (P.x-B.x)*a1;
        f0[1] = -(P.y-B.y)*a1;
        f1[2] = (P.x-C.x)*a2;
        f0[2] = -(P.y-C.y)*a2;
    }

    if (whatd[op_dx]) // value of the dx of function
    {
        assert(val.K()>op_dx);
        val(0,1,op_dx) = a0;
        val(1,1,op_dx) = a1;
        val(2,1,op_dx) = a2;
    }

    if (whatd[op_dy])
    {
        assert(val.K()>op_dy);
        val(0,0,op_dy) = -a0;
        val(1,0,op_dy) = -a1;
        val(2,0,op_dy) = -a2;
    }

    for (int i= op_dy; i< last_operatortype ; i++)
    if (whatd[op_dx])
        assert(op_dy);
}
```

The function to defined the coefficient $\alpha_k$:

```cpp
void TypeOfFE_RT::Pi_h_alpha(const baseFElement & K, KN_<double> & v) const
```
11.3. **HOW TO ADD**

```cpp
{  
    const Triangle & T(K.T);
    for (int i=0,k=0;i<3;i++)
    {
        R2 E(T.Edge(i));
        R signe = T.EdgeOrientation(i) ;
        v[k++]= signe*E.y;
        v[k++]= -signe*E.x;
    }
}
```

Now, we just need to add a new key work in FreeFem++, so at the end of the file, we add:

```cpp
static TypeOfFE_RTortho The_TypeOfFE_RTortho;
// let the 2 globals variables
static ListOfTFE typefemRTOrtho("RT0Ortho", & The_TypeOfFE_RTortho);
// ----- the name in freefem ----
// link with FreeFem++ do not work with static library .a
void init_FE_ADD() { }
```

To enforce in loading of this new finite element, we have to add the two new lines close to the end of files `src/femlib/FESpace.cpp` like:

```cpp
void init_static_FE()
{
    extern void init_FE_P2h() ;
    init_FE_P2h() ;
    extern void init_FE_ADD() ;
    init_FE_ADD() ;
}
```

### 11.3 How to add

First, create a file `FE_ADD.cpp` containing all this code, like in file `src/femlib/Element_P2h.cpp`, after modifying the `Makefile.am` by adding the name of your file to the variable `EXTRA_DIST` like:

```bash
# Makefile using Automake + Autoconf
# -----------------------------
# $Id: addfe.tex,v 1.3 2004/09/28 08:58:40 hecht Exp $

# This is not compiled as a separate library because its
# interconnections with other libraries have not been solved.
```
and recompile
For codewarrior compilation add the file in the project an remove the flag in panal PPC
linker FreeFEm++ Setting Dead-strip Static Initializition Code Flag.
Appendix A

Table of Notations

Here mathematical expressions and corresponding freefem++ commands are noted.

A.1 Generalities

δ_{ij}  Kronecker delta (0 if \( i \neq j \), 1 if \( i = j \) for integers \( i, j \))

\( \forall \) for all

\( \exists \) there exist

i.e. that is

PDE  partial differential equation (with boundary conditions)

\( \emptyset \) the empty set

\( \mathbb{N} \) the set of integers (\( a \in \mathbb{N} \leftrightarrow \text{int } a \)); “int” means long integer inside freefem++

\( \mathbb{R} \) the set of real numbers (\( a \in \mathbb{R} \leftrightarrow \text{real } a \)); double inside freefem++

\( \mathbb{C} \) the set of complex numbers (\( a \in \mathbb{C} \leftrightarrow \text{complex } a \)); complex\( \text{double} \)

\( \mathbb{R}^d \) d-dimensional Euclidean space

A.2 Sets, Mappings, Matrices, Vectors

Let \( E, F, G \) be three sets and \( A \) subset of \( E \).

\( \{ x \in E \mid P \} \) the subset of \( E \) consisting of the elements possessing the property \( P \)

\( E \cup F \) the set of elements belonging to \( E \) or \( F \)

\( E \cap F \) the set of elements belonging to \( E \) and \( F \)

\( E \setminus A \) the set \( \{ x \in E \mid x \notin A \} \)

\( E + F \) \( E \cup F \) with \( E \cap F = \emptyset \)
$E \times F$ the cartesian product of $E$ and $F$

$E^n$ the $n$-th power of $E$ ($E^2 = E \times E$, $E^n = E \times E^{n-1}$)

$f : E \to F$ the mapping form $E$ into $F$, i.e., $E \ni x \mapsto f(x) \in F$

$I_E$ or $I$ the identity mapping in $E$, i.e., $I(x) = x \ \forall x \in E$

$f \circ g$ for $f : F \to G$ and $g : E \to F$, $E \ni x \mapsto (f \circ g)(x) = f(g(x)) \in G$ (see Section 2.5)

$f|_A$ the restriction of $f : E \to F$ to the subset $A$ of $E$

$\{a_k\}$ column vector with components $a_k$

$(a_k)$ row vector with components $a_k$

$(a_k)^T$ denotes the transpose of a matrix $(a_k)$, and is $\{a_k\}$

$\{a_{ij}\}$ matrix with components $a_{ij}$, and $(a_{ij})^T = (a_{ji})$

A.3 Numbers

For two real numbers $a, b$

$[a, b] \{x \in \mathbb{R} \mid a \leq x \leq b\}$

$a, b] \{x \in \mathbb{R} \mid a < x \leq b\}$

$[a, b[ \{x \in \mathbb{R} \mid a \leq x < b\}$

$a, b[ \{x \in \mathbb{R} \mid a < x < b\}$

A.4 Differential Calculus

$\partial f/\partial x$ the partial derivative of $f : \mathbb{R}^d \to \mathbb{R}$ with respect to $x$ ($\mathbf{dx}(\xi)$)

$\nabla f$ the gradient of $f : \Omega \to \mathbb{R}$, i.e., $\nabla f = (\partial f/\partial x, \partial f/\partial y)$

$\text{div } f$ or $\nabla \cdot f$ the divergence of $f : \Omega \to \mathbb{R}^d$, i.e., $\text{div } f = \partial f_1/\partial x + \partial f_2/\partial y$

$\Delta f$ the Laplacian of $f : \Omega \to \mathbb{R}$, i.e., $\Delta f = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$
A.5 Meshes

Ω usually denotes a domain on which PDE is defined
Γ denotes the boundary of Ω, i.e., Γ = ∂Ω (keyword border, see Section 3.1.2)
$\mathcal{T}_h$ the triangulation of Ω, i.e., the set of triangles $T_k$, where $h$ stands for mesh size (keyword mesh, buildmesh, see Section 3)

$n_t$ the number of triangles in $\mathcal{T}_h$ (get by Th.nt, see “mesh.edp”)

$\Omega_h$ denotes the approximated domain $\Omega_h = \bigcup_{k=1}^{n_t} T_k$ of $\Omega$. If $\Omega$ is polygonal domain, then it will be $\Omega = \Omega_h$

$\Gamma_h$ the boundary of $\Omega_h$

$n_v$ the number of vertices in $\mathcal{T}_h$ (get by Th.nv)

$[q^i q^j]$ the segment connecting $q^i$ and $q^j$

$q^{k_1}, q^{k_2}, q^{k_3}$ the vertices of a triangle $T_k$ with anti-clock direction (get the coordinate of $q^{k_j}$ by $(\text{Th}[k-1][j-1].x, \text{Th}[k-1][j-1].y)$)

$I_\Omega$ the set $\{i \in \mathbb{N} | q^i \notin \Gamma_h\}$

A.6 Finite Element Spaces

$L^2(\Omega)$ the set $\left\{ w(x,y) \left| \int_{\Omega} |w(x,y)|^2 \, dx \, dy < \infty \right. \right\}$

norm: $\|w\|_{0,\Omega} = \left( \int_{\Omega} |w(x,y)|^2 \, dx \, dy \right)^{1/2}$

scalar product: $(v, w) = \int_{\Omega} vw$

$H^1(\Omega)$ the set $\left\{ w \in L^2(\Omega) \left| \int_{\Omega} \left( |\partial w/\partial x|^2 + |\partial w/\partial y|^2 \right) \, dx \, dy < \infty \right. \right\}$

norm: $\|w\|_{1,\Omega} = \left( \|w\|_{0,\Omega}^2 + \|\nabla w\|_{0,\Omega}^2 \right)^{1/2}$

$H^m(\Omega)$ the set $\left\{ w \in L^2(\Omega) \left| \int_{\Omega} \frac{\partial^{\alpha\beta} w}{\partial x^{\alpha_1} \partial y^{\alpha_2}} \in L^2(\Omega) \quad \forall \alpha = (\alpha_1, \alpha_2) \in \mathbb{N}^2, |\alpha| = \alpha_1 + \alpha_2 \right. \right\}$

scalar product: $(v, w)_{1,\Omega} = \sum_{|\alpha| \leq m} \int_{\Omega} D^\alpha v D^\alpha w$

$H^1_0(\Omega)$ the set $\{ w \in H^1(\Omega) \left| u = 0 \quad \text{on} \ \Gamma \}$
$L^2(\Omega)^2$ denotes $L^2(\Omega) \times L^2(\Omega)$, and also $H^1(\Omega)^2 = H^1(\Omega) \times H^1(\Omega)$.

$V_h$ denotes the finite element space created by \texttt{fespace Vh(Th,*)} in \texttt{freefem++} (see Section 4 for \texttt{*}).

$\Pi_h f$ the projection of the function $f$ into $V_h$ (\texttt{func f=x*y^3; Vh v = f;}) means $v = \Pi_h f$.

$\{v\}$ for FE-function $v$ in $V_h$ means the column vector $(v_1, \cdots, v_M)^T$ if $v = v_1 \phi_1 + \cdots + v_M \phi_M$, which is shown by \texttt{fespace Vh(Th,P2); Vh v; cout << v[] << endl;}. 


Appendix B

Grammar

B.1 Keywords

R3
bool
border
break
complex
continue
else
end
fespace
for
func
if
ifstream
include
int
load
macro
matrix
mesh
ofstream
problem
real
return
solve
string
varf
while

B.2 The bison grammar

\[
\text{start: input ENDOFILE;}
\]
APPENDIX B. GRAMMAR

input: instructions ;

instructions: instruction
  | instructions instruction ;

list_of_id_args:
  | id
  | id '=' no_comma_expr
  | FSPACE id
  | type_of_dcl id
  | type_of_dcl '& id
  | '[' list_of_id_args ']
  | list_of_id_args ']' id
  | list_of_id_args ']' '[' list_of_id_args ']
  | list_of_id_args ']' id '=' no_comma_expr
  | list_of_id_args ']' FSPACE id
  | list_of_id_args ']' type_of_dcl id
  | list_of_id_args ']' type_of_dcl '& id ;

list_of_id1: id
  | list_ofidi ']' id ;

id: ID | FSPACE ;

list_of_dcls: ID
  | ID '=' no_comma_expr
  | ID '(' parameters_list ')' 
  | list_of_dcls ']' list_of_dcls ;

parameters_list:
  no_set_expr
  | FSPACE ID
  | ID '=' no_set_expr
  | parameters_list ']' no_set_expr 
  | parameters_list ']' id '=' no_set_expr ;

type_of_dcl: TYPE 
  | TYPE '[' TYPE ']';

ID_space:
  ID
  | ID '[' no_set_expr ']
  | ID '=' no_set_expr
  | '[' list_of_id1 ']
  | '[' list_of_id1 ']' '[' no_set_expr ']
  | '[' list_of_id1 ']' '=' no_set_expr ;

ID_array_space:
  ID '[' no_set_expr ']
  | '[' list_of_id1 ']' '(' no_set_expr ')' ;

fespace: FSPACE ;

spaceIDa :
  | spaceIDa ']' ID_array_space ;
spaceIDb : ID_space
    | spaceIDb ',' ID_space ;

spaceIDs : fespace spaceIDb
    | fespace ' [' spaceIDa ;

fespace_def: ID ' (' parameters_list ') ' ;

fespace_def_list: fespace_def
    | fespace_def_list ',' fespace_def ;

declaration: type_of_dcl list_of_dcls ';' 
    | 'fespace' fespace_def_list ' ; '
    | spaceIDs ' ; '
    | FUNCTION ID '=' Expr ' ; '
    | FUNCTION type_of_dcl ID ' (' list_of_id_args ') ' '{' instructions ' } '
    | FUNCTION ID ' (' list_of_id_args ') ' '=' no_comma_expr ' ; '

begin: '{' ;
end: ' ) ' ;

for_loop: ' for ' ;
while_loop: ' while ' ;

instruction: ';' 
    | ' include' STRING 
    | ' load' STRING 
    | declaration 
    | for_loop ' (' Expr ';' Expr ';' Expr ')' instruction 
    | while_loop ' (' Expr ')' instruction 
    | ' if ' ' (' Expr ')' instruction 
    | ' if ' ' (' Expr ')' instruction ELSE instruction 
    | begin instructions end 
    | ' border' ID border_expr 
    | ' border' ID ' [ ' array ' ] ' ' ; 
    | ' break ' ' ; 
    | ' continue ' ' ; 
    | ' return' Expr ' ; ' ;

bornes: ' ( ' ID ' = ' Expr ' , ' Expr ' ) ' ;

border_expr: bornes instruction ;

Expr: no_comma_expr
    | Expr ' , ' Expr ;

unop: '-' 
    | '+' 
    | ' !' 
    | ' ++ ' 
    | ' -- ' ;
APPENDIX B. GRAMMAR

no_comma_expr:
  no_set_expr
  | no_set_expr '==' no_comma_expr
  | no_set_expr '+=' no_comma_expr
  | no_set_expr '-=' no_comma_expr
  | no_set_expr '*=' no_comma_expr
  | no_set_expr '/=' no_comma_expr ;

no_set_expr:
  unary_expr
  | no_set_expr '*' no_set_expr
  | no_set_expr '.' no_set_expr
  | no_set_expr '/' no_set_expr
  | no_set_expr '%' no_set_expr
  | no_set_expr '+' no_set_expr
  | no_set_expr '-' no_set_expr
  | no_set_expr '<<' no_set_expr
  | no_set_expr '>>' no_set_expr
  | no_set_expr '&' no_set_expr
  | no_set_expr '&&' no_set_expr
  | no_set_expr '|' no_set_expr
  | no_set_expr '||' no_set_expr
  | no_set_expr '<' no_set_expr
  | no_set_expr '<=' no_set_expr
  | no_set_expr '>' no_set_expr
  | no_set_expr '>=' no_set_expr
  | no_set_expr '==' no_set_expr
  | no_set_expr '!=' no_set_expr ;

parameters:
  no_set_expr
  | FESPACE
  | id '==' no_set_expr
  | parameters ',' FESPACE
  | parameters ',' no_set_expr
  | parameters ',' id '==' no_set_expr ;

array:
  no_comma_expr
  | array ',' no_comma_expr ;

unary_expr:
  pow_expr
  | unop pow_expr %prec UNARY ;

pow_expr:
  primary
  | primary '^' unary_expr
  | primary '_' unary_expr
  | primary ' ´' ; // transpose

primary:
  ID
  | LNUM
The Types of the languages, and cast

the types

--lgElement = <lgElement>
  [, type :<Polymorphic>
    operator : ( <lgVertex> : <lgElement>, <long> )

--lgVertex = <lgVertex>
  label, type :<Polymorphic>
    operator. : ( <long> : <lgVertex> )

    x, type :<Polymorphic>
    operator. : ( <double> : <lgVertex> )

    y, type :<Polymorphic>
    operator. : ( <double> : <lgVertex> )

--Add_KN_<double> = <Add_KN_<double>>

--Add_Mulc_KN_<double> * = <Add_Mulc_KN_<double>>

--AnyTypeWithOutCheck = <AnyTypeWithOutCheck>

--C_F0 = <C_F0>

--DotStar_KN_<double> = <DotStar_KN_<double>>

--E_Array = <E_Array>
--FEbase<double> * = <FEbase<double>>
  <FEbase<double>> : <FEbase<double>>
--FEbase<double> ** = <FEbase<double> **>

--FEbaseArray<double> * = <FEbaseArray<double>>
--FEbaseArray<double> ** = <FEbaseArray<double> **>
  [] type :<Polymorphic> operator :
  ( <FEbase<double> **> : <FEbaseArray<double> **>, <long> )

--Fem2D::Mesh * = <Fem2D::Mesh>
  <Fem2D::Mesh> : <Fem2D::Mesh **>
--Fem2D::Mesh ** = <Fem2D::Mesh **>
  <-, type :<Polymorphic>
  ( <Fem2D::Mesh> : <string> )
  ( <long> : <Fem2D::Mesh **>, <double>, <double> )

  area, type :<Polymorphic> operator. :
  ( <double> : <Fem2D::Mesh **> )

  nt, type :<Polymorphic>
  operator. :
  ( <long> : <Fem2D::Mesh **> )

  nv, type :<Polymorphic> operator. :
  ( <long> : <Fem2D::Mesh **> )

--Fem2D::MeshPoint * = <Fem2D::MeshPoint>
  N, type :<Polymorphic> operator. :
  ( <Fem2D::R3> : <Fem2D::MeshPoint> )

  P, type :<Polymorphic> operator. :
  ( <Fem2D::R3> : <Fem2D::MeshPoint> )

--Fem2D::R2 * = <Fem2D::R2>
--Fem2D::R3 * = <Fem2D::R3>
  x, type :<Polymorphic> operator. :
  ( <double *> : <Fem2D::R3> )

  y, type :<Polymorphic> operator. :
  ( <double *> : <Fem2D::R3> )

  z, type :<Polymorphic> operator. :
B.3. THE TYPES OF THE LANGUAGES, AND CAST

( <double *> : <Fem2D::R3> )

--Fem2D::TypeOfFE * = <Fem2D::TypeOfFE>

--KN<double> = <KN<double>>

[] type :<Polymorphic> operator :
  ( <double *> : <KN<double>>, <long> )

--KN<double> * = <KN<double> *>

<-, type :<Polymorphic>
  ( <KN<double> *> : <KN<double> *>, <long> )

[] type :<Polymorphic> operator :
  ( <double *> : <KN<double> *>, <long> )

max, type :<Polymorphic> operator. :
  ( <double > : <KN<double> *> )

min, type :<Polymorphic> operator. :
  ( <double > : <KN<double> *> )

n, type :<Polymorphic>
operator. :
  ( <long > : <KN<double> *> )

sum, type :<Polymorphic> operator. :
  ( <double > : <KN<double> *> )

--KN_<double> = <KN_<double>>

--KN_<double> * = <KN_<double> *>

--Matrice_Creuse<double> * = <Matrice_Creuse<double>>

<Matrice_Creuse<double>> : <Problem>

--Matrice_Creuse_Transpose<double> = <Matrice_Creuse_Transpose<double>>

--Matrice_Creuse_inv<double> = <Matrice_Creuse_inv<double>>

--Mulc_KN_<double> = <Mulc_KN_<double>>

--MyMap<String, double> * = <MyMap<String, double>>

[] type :<Polymorphic> operator :
  ( <double *> : <MyMap<String, double>>, <string> )
APPENDIX B. GRAMMAR

--Polymorphic * = <Polymorphic>
--Sub_KN_<double> = <Sub_KN_<double>>
--Transpose<KN<double>> = <Transpose<KN<double>>>
--TypeSolveMat * = <TypeSolveMat>
--VirtualMatrice<double>::plusAtx = <VirtualMatrice<double>::plusAtx>
--VirtualMatrice<double>::plusAx = <VirtualMatrice<double>::plusAx>
--VirtualMatrice<double>::solveAxeqb = <VirtualMatrice<double>::solveAxeqb>
--bool = <bool>
  <bool> : <bool *>
--bool * = <bool *>
--char * = <char>
--const BC_set<double> * = <BC_set<double>>

--const CDomainOfIntegration * = <CDomainOfIntegration>
  () type :<Polymorphic> operator :
  ( <FormBilinear> : <CDomainOfIntegration>, <LinearComb<std::pair<MGauche, MDroit>, C_F0>> )
  ( <double> : <CDomainOfIntegration>, <double> )
  ( <FormLinear> : <CDomainOfIntegration>, <LinearComb<MDroit, C_F0>> )

--const C_args * = <C_args>
  <C_args> : <FormBilinear> () type :<Polymorphic> operator :
  ( <Call_FormLinear> : <C_args>, <long>, <v_fes **> )
  ( <Call_FormBilinear> : <C_args>, <v_fes **>, <v_fes **> )

--const Call_FormBilinear * = <Call_FormBilinear>
--const Call_FormLinear * = <Call_FormLinear>
--const E_Border * = <E_Border>
--const E_BorderN * = <E_BorderN>
--const Fem2D::QuadratureFormular * = <Fem2D::QuadratureFormular>
--const Fem2D::QuadratureFormular1d * = <Fem2D::QuadratureFormular1d>
--const FormBilinear * = <FormBilinear>
   () type :<Polymorphic> operator :
   (   <Call_FormBilinear> : <FormBilinear>, <v_fes **>, <v_fes **> )
   (   <Call_FormLinear> : <FormBilinear>, <long>, <v_fes **> )

--const FormLinear * = <FormLinear>
   () type :<Polymorphic> operator :
   (   <Call_FormLinear> : <FormLinear>, <v_fes **> )

--const IntFunction * = <IntFunction>

--const LinearComb<MDroit, C_F0> * = <LinearComb<MDroit, C_F0>>
--const LinearComb<MGauche, C_F0> * = <LinearComb<MGauche, C_F0>>
--const LinearComb<std::pair<MGauche, MDroit>, C_F0> * = <LinearComb<std::pair<MGauche, MDroit>, C_F0>>

--const Problem * = <Problem>

--const Solve * = <Solve>

--const char * = <char>

--double = <double>
   <double> : <double *> () type :<Polymorphic> operator :
   (   <double> : <double>, <double>, <double> )

--double * = <double *>

--interpolate_f_X_1<double>::type = <interpolate_f_X_1<double>::type>

--long = <long>
   <long> : <long *>
--long * = <long *>

--istream * = <istream>
   <istream> : <istream **>
--istream ** = <istream **>

--ostream * = <ostream>
   <ostream> : <ostream **>
--ostream ** = <ostream **>
   <-, type :<Polymorphic> operator( ):
B.4 All the operators

- CG, type :<TypeSolveMat>
- Cholesky, type :<TypeSolveMat>
- Crout, type :<TypeSolveMat>
- GMRES, type :<TypeSolveMat>
- LU, type :<TypeSolveMat>
- LinearCG, type :<Polymorphic> operator() :
  (  <long> :  <Polymorphic>,  <KN<double> *>,  <KN<double> *> )

- N, type :<Fem2D::R3>
- NoUseOfWait, type :<bool *>
- \( P \), type :\( \text{Fem2D::R3} \)
- \( P_0 \), type :\( \text{Fem2D::TypeOfFE} \)
- \( P_1 \), type :\( \text{Fem2D::TypeOfFE} \)
- \( P_{1nc} \), type :\( \text{Fem2D::TypeOfFE} \)
- \( P_2 \), type :\( \text{Fem2D::TypeOfFE} \)
- \( RT_0 \), type :\( \text{Fem2D::TypeOfFE} \)
- \( RT_{modif} \), type :\( \text{Fem2D::TypeOfFE} \)
- \( \text{abs} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{acos} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{acosh} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{adaptmesh} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{Fem2D::Mesh} \) : \( \text{Fem2D::Mesh} \)... )

- \( \text{append} \), type :\( \text{std::ios_base::openmode} \)
- \( \text{asin} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{asinh} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{atan} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{atan2} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{atanh} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{double} \) : \( \text{double} \) )

- \( \text{buildmesh} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{Fem2D::Mesh} \) : \( \text{E_BorderN} \) )

- \( \text{buildmeshborder} \), type :\( \text{Polymorphic} \) operator() :
  ( \( \text{Fem2D::Mesh} \) : \( \text{E_BorderN} \) )

- \( \text{cin} \), type :\( \text{istream} \)
- \( \text{clock} \), type :\( \text{Polymorphic} \)
  ( \( \text{double} \) : \( \) )

- \( \text{conj} \), type :\( \text{Polymorphic} \) operator() :
APPENDIX B. GRAMMAR

- convect, type :<Polymorphic> operator() :
  ( <double> : <E_Array>, <double>, <double> )

- cos, type :<Polymorphic> operator() :
  ( <double> : <double> )
  ( <complex> : <complex> )

- cosh, type :<Polymorphic> operator() :
  ( <double> : <double> )
  ( <complex> : <complex> )

- cout, type :<ostream>

- dumptable, type :<Polymorphic> operator() :
  ( <ostream> : <ostream> )

- dx, type :<Polymorphic> operator() :
  ( <LinearComb<MDroit, C_F0>> : <LinearComb<MDroit, C_F0>> )
  ( <double> : <std::pair<FEbase<double> *, int>> )
  ( <LinearComb<M_Gauche, C_F0>> : <LinearComb<M_Gauche, C_F0>> )

- dy, type :<Polymorphic> operator() :
  ( <LinearComb<MDroit, C_F0>> : <LinearComb<MDroit, C_F0>> )
  ( <double> : <std::pair<FEbase<double> *, int>> )
  ( <LinearComb<M_Gauche, C_F0>> : <LinearComb<M_Gauche, C_F0>> )

- endl, type :<char>

- exec, type :<Polymorphic> operator() :
  ( <long> : <string> )

- exit, type :<Polymorphic> operator() :
  ( <long> : <long> )

- exp, type :<Polymorphic> operator() :
  ( <double> : <double> )
  ( <complex> : <complex> )

- false, type :<bool>

- imag, type :<Polymorphic> operator() :
  ( <double> : <complex> )

- int1d, type :<Polymorphic> operator() :
  ( <CDomainOfIntegration> : <Fem2D::Mesh>... )

- int2d, type :<Polymorphic> operator() :
  ( <CDomainOfIntegration> : <Fem2D::Mesh>... )
B.4. ALL THE OPERATORS

- intalledges, type :<Polymorphic>
  operator( :  
    (  <CDomainOfIntegration> :  <Fem2D::Mesh>... )

- jump, type :<Polymorphic>
  operator( :  
    (  <LinearComb<MDroit, C_F0>> :  <LinearComb<MDroit, C_F0>>
    (  <double> :  <double> )
    (  <LinearComb<MGauche, C_F0>> :  <LinearComb<MGauche, C_F0>>

- label, type :<long *>

- log, type :<Polymorphic>  operator() :
  (  <double> :  <double> )
  (  <complex> :  <complex> )

- log10, type :<Polymorphic>  operator() :
  (  <double> :  <double> )

- max, type :<Polymorphic>  operator() :
  (  <double> :  <double>, <double> )
  (  <long> :  <long>, <long> )

- mean, type :<Polymorphic>
  operator( :  
    (  <double> :  <double> )

- min, type :<Polymorphic>  operator() :
  (  <double> :  <double>, <double> )
  (  <long> :  <long>, <long> )

- movemesh, type :<Polymorphic>  operator() :
  (  <Fem2D::Mesh> :  <Fem2D::Mesh>, <E_Array>... )

- norm, type :<Polymorphic>
  operator( :  
    (  <double> :  <std::complex<double>> )

- nuTriangle, type :<long>
- nuEdge, type :<long>
- on, type :<Polymorphic>  operator() :
  (  <BC_set<double>> :  <long>... )

- otherside, type :<Polymorphic>
  operator( :  
    (  <LinearComb<MDroit, C_F0>> :  <LinearComb<MDroit, C_F0>>
    (  <LinearComb<MGauche, C_F0>> :  <LinearComb<MGauche, C_F0>> )


- pi, type :<double>
- plot, type :<Polymorphic> operator() :
  (  <long> : ... )

- pow, type :<Polymorphic> operator() :
  (  <double> : <double>, <double> )
  (  <complex> : <complex>, <complex> )

- qf1pE, type :<Fem2D::QuadratureFormular1d>
- qf1pT, type :<Fem2D::QuadratureFormular>
- qf1pTlump, type :<Fem2D::QuadratureFormular>
- qf2pE, type :<Fem2D::QuadratureFormular1d>
- qf2pT, type :<Fem2D::QuadratureFormular>
- qf2pT4P1, type :<Fem2D::QuadratureFormular>
- qf3pE, type :<Fem2D::QuadratureFormular1d>
- qf5pT, type :<Fem2D::QuadratureFormular>

- readmesh, type :<Polymorphic> operator() :
  (  <Fem2D::Mesh> : <string> )

- real, type :<Polymorphic> operator() :
  (  <double> : <complex> )

- region, type :<long *>
- savemesh, type :<Polymorphic> operator() :
  (  <Fem2D::Mesh> : <Fem2D::Mesh>, <string>... )

- sin, type :<Polymorphic> operator() :
  (  <double> : <double> )
  (  <complex> : <complex> )

- sinh, type :<Polymorphic> operator() :
  (  <double> : <double> )
  (  <complex> : <complex> )

- sqrt, type :<Polymorphic> operator() :
  (  <double> : <double> )
  (  <complex> : <complex> )

- square, type :<Polymorphic> operator() :
  (  <Fem2D::Mesh> : <long>, <long> )
  (  <Fem2D::Mesh> : <long>, <long>, <E_Array> )

- tan, type :<Polymorphic> operator() :
  (  <double> : <double> )
B.5 History of the software

beginning: november 21, 2001: version 1.08.
----------------------------------------------
2001/11/22: correction of operator == and !=
2001/11/23, version: 1.09: correction (with g++)
template<class A> struct SameType, type of OK must be int and not bool
2001/11/24 add fonctionality in plot, bb=[[x1,y1],[x2,y2]]
add loop if enter character +,-,=,c,C,r in graphic window.
2001/11/28
  correction bug initialization of QuadTree if less than 4 points in the quadtree files QuadTree.cpp and FQuadTree.hpp
  add exec("xxx..."); // to execute on system command. "xxx ...
  add dumptable(cout);  // to show all internal table
2001/11/29:
  Version ans graphique + ajout d’option dans plot (cf. doc)
correction ajoute renum() des maillage crees.
2001/12/10
  Correct missing check in plot
    ex: plot(1); trap before now, genere a compile error
  Correct in the interpolation is full not conforme FE,
do not prolonged by continuity. change
  add after line 479 in file lgfem.cpp
    if (outside && !KK.tfe->NbDfOnVertex && !KK.tfe->NbDfOnEdge)
      return SetAny<R>(0.0);
2001/12/12
  correction in gibbs (mesh renum) reconstruct the array of
triangle for each vertex (PB. of interpolation in non
convexe domain) Big bug.

Make version 1.14

----------

2001/12/14
  correction in trunc mesh, bug if empty mesh is created,
and move a little the test point in a triangle
is not exactly the barycenter.

2002/01/14
  correction in probem.cpp line 1309 bug when we write qft= ... , in int2d
if (nargs[0]) return *GetAny<const Fem2D::QuadratureFormular*>(nargs[1])...
becomes:
if (nargs[0]) return *GetAny<const Fem2D::QuadratureFormular*>(nargs[0])...

2002/01/15
in lgfem.cpp remove line 522 reffecran(); // bug if not graphique some time
in file MeshGeom.cpp line 108 bug if name == 0 ; add test before the cout like
if(name) cout << " ... " << name << ....
correct the name of the exec file (FreeFee++ -> FreeFem++) in Makefile

2002/02/03
in Mesh2.cpp in preinit() add call to srand(),
to get the same mesh with the same data.
correct
mesh tth=th;

Make version 1.15
-----------------

2002/02/20
Add periodic boundary condition see the manual //

Make version 1.16
-----------------

Add Parallele Mpi
correct bug in SegmentationFault.edp missing placing of delete[] operator
put declaration in for
wait =xx change the default value of wait
ofstream f("foo.txt",append) open a file in append mode
add NoUseOfWait=true; never wait for run test easilyNoUseOfWait

Make version 1.17 2002/03/20
----------------------------

Make default iso value changing with zoom option.
Add GMRES solver  dimKrylov= , tgv= //

april 2002
Make the current version without CheckPtr, and
correct some bug in string allocation (forgetting +1 in some length)
Improving the speed of the software
add CPUTime global bool variable to print the CPU time of each instruction.
correct some printing without verbosity
adding some checking in array management
Add option in the Makefile (gnumake)

Make version 1.18 the 2002/04/08
-------------------------------

Big correction in automatic cast see bugv1.18.edp
correction in interpolation of label see also bugv1.18.edp
Big correction in construction of non-symetrix matrix (see BUG)

Make version 1.19 2002/04/18
B.5. HISTORY OF THE SOFTWARE

---------------

Correction in Quadtree integer overflow when interpolate solution of from one big domain to a very small one.

Correction in embedded function (return problem)
Correction in return type of function (right value an note left value)
Correction cast bool to int, so lots of bug in expression like \((\text{region}==2)\times5\) always 0, invisible before 1.18 (because bug in automatic cast)

Make version 1.20 2002/04/25
---------------

Correction of bug in interpolation on non convexe domain.
Correction of bug in periodic boundary condition (pb of sens in french)
Correction some compilation bug under g++ v3.0 in RNM file
Correction find common point buildmesh (change threshold value)

Make version 1.21 2002/04/29
---------------

Add Non linear Conjuged Gradient (CGNL) routine.
correct bug in linearGC with non zero right and size.

Make version 1.22 2002/05/02
---------------

all optimisation tool from // http://coool.mines.edu/
Add BFGS optimisation tools form cool //
correct bug in CGNL, correct tgv=, in linear form //
make algo.edp example

Make version 1.23 2002/05/13
---------------

correct bug of the symetric matrix are independ of \(x, y, \ldots\)
add line at 285, in file problem.cpp:
    MeshPointStack(stack)->set(T(pi),pi,Ku);

correct bug in sign of int1d in linearform,
    so change in exemple fluidstructure, schwarz-no-overlap, LaplaceP1,
    aalapacien, lapacienprecon
correct some bug in -= operator with result of \(\ast. \ast\)
add some operators:
    -square(u) = u^2
    -intalledges(Th)(...) to compute integrale on all edges of all triangle //

for error indicator add global :
    -lenEdge the len of the current edge
    -hTriangle the size of the current triangle
    -area the area of the current triangle

Make version 1.24
---------------
add in BamgFreeFem.cpp after line 211 to set name
Tn->name= new char[strlen("msh2bamg")]+1];
strcpy(Tn->name,"msh2bamg");
add – of linearform and bilinearform see LaplaceP1bis.edp

correct: July 8, 2002
Vh u1,u2,v1,v2;
bug in problem(u1,u2,v1,v2,...) =
in file: FESSpace.hpp line 193:
{ throwassert(dim_which_sub_fem[N-1]]>=0 && dim_which_sub_fem[N-1]< nb_sub_fem);
for(int i=0,n0=0,l=0,i0=0; i<k; i++,n0+=t.N,i0+=t.NbDoF)
   for(int j=0;j<t.pij_alpha.N();j++,l++) {
      pij_alpha[l].i=t.pij_alpha[j].i+i0;
      pij_alpha[l].p=t.pij_alpha[j].p;
      pij_alpha[l].j=t.pij_alpha[j].j+n0;
   }
}
add plot of list of border //
add init array:
real[int] a=[1,2,3,5.6,7,8,9]; /*
Make version 1.25
-----------------
13 aout 2002 Major Bug in : FESpace.hpp ligne 199
All the non scalar problem with same kind of finite element do not work:
FESpace.hpp ligne 199
   for(int i=0,n0=0,l=0,i0=0; i<k; i++,n0+=t.N,i0+=t.NbDoF)
      for (int j=0;j<t.pij_alpha.N();j++,l++) {
         pij_alpha[l].i=t.pij_alpha[j].i+i0;
         pij_alpha[l].p=t.pij_alpha[j].p;
         pij_alpha[l].j=t.pij_alpha[j].j+n0;
      }
become
   // Warning the component is moving first
   for (int j=0,l=0;j<t.pij_alpha.N();j++) // for all sub DF
      for(int i=0,i0=0; i<k; i++,l++) // for component
         {
            pij_alpha[l].i=t.pij_alpha[j].i+k+i; // DoF number
            pij_alpha[l].p=t.pij_alpha[j].p; // point of interpolation
            pij_alpha[l].j=t.pij_alpha[j].j+i*t.N; // component of interpolation
         }
   // Warning the component is moving first
   //
two corrections in problem.cpp :
   2 missing delete (example laplacienprecon.edp)
   2 delete -> delete [] (example testFE.edp)

26 septembre 2002
add New Finite element RTortho (a conforme //
FE in $H(\text{curl})$ like RT conforme FE in $H(\text{div})$.
see exemple aaRT.epd

correct the computation of $dx(v)$ and $dy(u)$ in $[u,v]$ RT finite element function.
compile on hpux 11 with gcc 3.2 see Makefile-hp9000s700

correct argument passing in GMRES routine

Make version 1.26 le 26 septembre 2002

05 nov. 2002 Correct very small mistake in bamglib part.

19 nov 2002:
add:
- tanh function
- qf2pT4P1 a triangular QuadratureFormular
(4P1 the qf2pT QuadratureFormular)
small change une meshadapt to just compute the metrix see exemple
convectapt.edp
make interpolation matrix between to FEspace (not wet finish)

3/12/2002
make correction to get a not to bad linenumber in error.
add add test is used of unset x,y to make a error.
add checkmovemesh(Th,[x+u,y+v]) function to return the
value of the area of the minimal triangle of the movemesh.
see mesh.edp example.

23/12/2002:
add eigen value solver
see eigen README_ARPACK to compile
see examples++-eigen to test.

Make version 1.28 2/1/2003

09/1/2003:
add sub array option in array:
real[int] tab(100);
tab(:) = array from 0 to 99=tab.n-1
tab(2:10) sub array from 2 to 10
tab(2:10:2) sub array from 2 to 10 by step 2
tab(2:10:2)[0] = 5 ; => modification of tab[2] = 5;
tab(2:10:2)[1] = 6 ; => modification of tab[2+2] = 6;

16/01/2003
add macro generation like cpp preprosseur:
this is usefull to make automatic diff
exemple:
real cc=2;
macro f(u) (cc*(exp(u)-1))
macro df(f) (cc*(exp(f)))
real u=1;
cout << (cc*(exp(u)-1)) << endl;
cout << f(u) << endl;
cout << df(f(u)) << endl;
see macro.edp for more detail:

22/01/2003
in file lex.cpp add .c_str() for compilation problem on g++ 2.95

15/04/2003
add new finite element:
P1b  P1 + Bubble  //
P1dc P1 discontinious  //
P2dc P2 discontinious  //
correct bug periodic BC with vectorial FESpace, add the code.

Make version 1.31
-----------------

23/04/2003
small correction to compile with g++ 2.95.2 in eigen value tools

26/04/2003
add function triangulate(filename) to build the Delaunay Triangulation of a set of points in R^2. to build a function form a set of : x y f(x,y) see exemple mesh.edp

make version 1.32 (29/04/2003)
-----------------

- add optimization in automatique interpolation
CPU of all tutorial exemples 172 s (mon my Mac) new version 1.33 (with graphics)
CPU of all tutorial exemples 167 s (mon my Mac) new version 1.33 (without graphics)
CPU of all tutorial exemples 169 s (mon my Mac) new version 1.32 (without graphics) bofbof.

-correct bug in macro generation, a macro existe just in a block {..}.

- small correction in parallelempi.cpp add a ; line 159

Make version 1.33 01/07/2003
-----------------------------
- small correction in parallelempi.cpp add a ; line 159
- comment line 519 in file AFunction.hpp (pb with g++ 3.3.1)
  // operator Expression() const return f;
(double definition)see line 527
21/08/2003:
- Bug in optimization in case on non constant robin boundary condition
  correct
- add grey=1 ion plot command to make grey plot, and
  //
  correct small mistake (on mac and X11, ...)

Make version 1.34 (22/08/2003)
-----------------------------

16/09/2003
- bug in local variable stack if the size is larger the 8 (complex)
  complex a; real b;
  a=0; b=1;
  // here a is 0+i;

correction is:
  voila la correction dans AFunction.hpp vers la ligne 1486:
  template<class T>
  inline Type_Expr NewVariable(aType t, size_t & off)
  {     
      size_t o = align8(off);
      // align
      // off += t->un->ptr_type->size;
      // bug off += t->size;
      off += t->un_ptr_type->size;
      // correction
      return Type_Expr(t, new T(o, t));
  }

02/11/2003
- correct probleme of renumbering the triangles of a mesh
  when reading in the bamg software (with readmesh command).
  This implies error when we restore mesh and non P1 finite element
  solution, see example saveandrestore.edp.

28/11/2003
- correction of bug in window version
  add the umfpack sparse linear solver (this solver have some problem
  in some case), so I do’nt put this one in default linear solver.
  remark, I think is due to the way of taking Dirichlet boundary condition
  Huge value on diagonal (tgv=1e30) and Stokes matrice.
  //

- put this file on the web
- put on the web
- the codewaroir projet to build arpack ands umfpack lib.

Make version 1.36
-----------------

- Change the graphic window in Window xx version

8/12/2003
- reparer in mpi version get mesh via mpi
- forget build quadtree soo the interpolation bugs
- add 4 line in file parallelempi.cpp in mesh serialization (recivied
    // add 3 line FH 08/12/2003 forget build quadtree sorry
    Fem2D::R2 Pn,Px;
    a->BoundingBox(Pn,Px);
    a->quadtree=new FQuadTree(m,Pn,Px,m->nv);

18/12/2003
- add in mpi version: //
  broadcast(processor(1),th); // broadcast th from proc 1 to all other.
  see the mpi exemple
  change the precision to 6 to 12 is savemesh.
- add tool to change the default precision on ostream or ofstream
  with the C++ syntaxe
  see saverestore.edp example.
  cout.precision(12); //
- add UMFPACK linear solver ( not well test)
  http:  // www.cise.ufl.edu/research/sparse/umfpack
- add full matrix with few operator
  real [int,int] A(10,10);
  A= 2;
  A(5,5) = 2;
  cout << A << endl;
- add array of mesh

08/01/2004
- merge all mesh example in one file call mesh.edp
- add discontinuous Galerkin method (see LapDG2.edp example)
- add syntaxe to get mesh information (see end of mesh.edp exemple)
- add meshsplit function to make confromal recusive locat mesh splitting //
  (see mesh.edp example)
- add dynamic load via dlopen see load.edp in example++-load //

Make version 1.37
-------------

15/01/2004
- change metrix= in metric= the in adapted mesh metric=[m11,m12,m22] //
  (change the ordre to compatible with the fonction order with IsMetric=1)

02/02/2004
- change printing in gmres algorithme
- print the size of matrix

06/02/2004
- change intalledges in change of discontinuous Galerkin loop also boundary edge //
  jump(u) is external - internal value of u with normal go to internal to external
  jump(u) on boundary is -internal value of u //
  average(u) on boundary is internal value of u //
- add nTonEdge to get the number of Triangles which see the current edge //

so see the new LapDG2.edp example for full detail

07/02/2004
- correct bug in real[string] map;
  // map array
  change in get_element and operator < of String
- add int array

Make Version 1.38

--------------------
- remove (void) line 14 of throwassert.hpp file (erreur compile g++3.3.5)
make change in macro expansion of shell (no perfect to day)

16/04/2004
- add matrix tools
  bluid interpolation interpolation matric from a FEspace VH to an other FEspace Vh
  
  matrix Ih=matrix B= interplotematrix(VH,Vh);
  // where Vh correpond to line and VH to column

  the named parameters can be
  t= true or false (to bluid the transpose or not matrix)
  op=0 ,1 or 2 (to build the interpolation of value, dx,dy )
  inside=true or false (to remove or not all ouside interpolation quadrature point)
  build diagonal sparce matrix (type is morse)
  do the matrice product of to sparce matrix (type is morse)
  add build a sparce matrix from a full matrix so if you have install UMFPACK solver
  it is possible to solve systeme with full matrix.
  add set function to change the solver
  see exemple sparce-matrix.edp

- correct bug flag ouside of mesh
  add line in file fem.cpp line 864
  outside=true;
  so some time the P0 interpolatation is no zero
  add new king of mesh of multiplicator data:
  mesh emptyTh=emptymesh(Th);  // to build a mesh with no internal point
  see mesh.edp exemple

Make Version 1.40

--------------------

- add les linear combination of sparce matrice
- set a sparce solver to a sparce matrice (to day just UMFPACK)
- transforme a full matrix in sparce matrix
- add istream f("toto");
  f.good() or f.EOF to test state the file
  remark: the state is set after the read so the previous value is wrong

02/06/2004

--------
- correction in call macro with string "..." parameter

24/06/2004

--------
add tools to bluid periodic adampesh see sphere.edp of exemple
add HaveUMFPACK global variable see sparse-matrix.edp exemple
Change all the structure of the software
Use ./configure to build all the make file see (README)
you can build the Window version with g++ under cygwin
( http: cygwin.... )

Make Version 1.41

--------------------
an anonymous CVS server is available to get FreeFem++ source
cvs -d :pserver:anonymous@idared.ann.jussieu.fr:/Users/pubcvs/cvs login
(password = freefem++)
cvs -d :pserver:anonymous@idared.ann.jussieu.fr:/Users/pubcvs/cvs freefem++ co
the TAG release_1_41_before_packaging is created

30/08/04
---------
Correct divide by 0 in plot instruction when the bounding box is flat.
Set default solver to LU because UMFPACK sometime by bad result.
Correct all g++-3.4 error
Optimize Choleski and Crout solver (divide cpu time by two on pentium)
Add stuff for adding new finite element due to the new software architecture.

31/08/04
---------
Add string[string] array
correct bug in macro expansion
macro parameter will be not expanded
example of bug:
  macro a(i) i
  macro b a(x)
  b = 1 ;
  // we get "i=1" or we want "x=1"


Make Version 1.42
-----------------
11/10/04
---------
Add Finite element with complex value, and matrix<complex>
see sparse-cmatrix.edp and FEComplex.edp files
for a full examples
Vh<complex> ur;
varf av(u,v)=int(Th)(u*v*(1i+1));
Matrix<complex> A = vav(Vh,Vh);
  // warning don’t mixte Matrix<complex> and Matrix<real> in a expression

19/10/04
---------
correct UMFPACK driver problem, ands set this solver as a default
solver.
add finite element connectivite inquire ice FE.epd exemple
fespace Wh(Th,P2);
Wh.nt          // given the number of element
Wh.ndof        // given the number of degree of freedom
Wh.ndofK       // given the number of degree of freedom on one element
Wh(k,i)        // given the number of i degree of freedom of element k.
                  // with 0 <= i < Wh.ndofK and 0 <= k < Wh.nt

build the source and compile version 1.43
01/11/04
correct problem on period adapted mesh if orientation are reverse.
add cvs tag release_1_43
Now the current version is 1.44 (CVS)

04/11/004
--------
correct problem in LinearGMRES and LinearGC
in LinearCG algorithm for solving A*x = b0 + b
the matrix function can compute A*x or A*x - b0,
and the other part b of the right end side can exist or not.
But in LinearGMRES must the function must compute A*x and
b0 can't exist.
see algo.edp exist for a full example

06/12/2004
change function name interplotematrix to interpolate
correct bug introduce in 04/12 change
a new graphic FreeFem++-cs: integrated development environment (located in src/ide)
see README and manual-full.pdf
add new examples++-tutorial
mat_interpol.edp # to test interpolate function
shur-comp.edp # a shur complement domaine decomposition

11/12/2004
TAG release_1_44
now the current version is 1.45

29/12/2004
add concat operator # in macro generator.

04/01/2005
add new quadrature formular on edge (a,b) :qf1pElump : 1/2(f(a)+f(b))*length(ab)

05/01/2005
correction bug unset FE variable dans le .edp =>
use a null pointeur (in sol) to set the *X array in problem.cpp
(*X)[K(df)] = (*sol[kfe])[SK(kdf)];
make some time make a seg trap error.

06/01/2005
rewrite the macro generation tool to be more to suppress some bug
in argument substitution
correct error in line numbering in case of newline in string
cosmetic change in output

20/01/2005
count () in macro parameter to handle couple of () in macro parameter.
macro sum(a,b) (a+b);
sum((1),2) // now works and given (1)+2
sum(atan2(2,1),2) // now works and given atan2(2,1)+2

correct bug in linear combination of sparse matrix:
the correction is:
change line 1069, in file MatriceCreuse_tpl.hpp
mij[make_pair(i,j)] = coef*a[k];
to
    mij[make_pair(i,j)] += coef*a[k];

Correct a segfault if second derivative of test function exist in
problem definition. The Change in problem.cpp:
resize : KN<double> buf(Vh.MaximalNbOfDF() * 3 * Vh.N);
to : KN<double> buf(Vh.MaximalNbOfDF() * last_operator_type * Vh.N);

24/01/2005
add named parameter nbtx=.. in buildmesh function to change the default
maximal number of vertex in the build mesh.

26/01/2005
correct pb with --disable-pdf in configure.ac
correct pdflatex and latex compilation
correct FreeFem++.app probleme when usind in a shell script FreeFem++-CoCoa
Bibliography


[8] Modulef ??????????


197


[20] Bison documentation


[22] COOOL: a package of tools for writing optimization code and solving optimization problems,


[24] COOL packagehttp://coool.mines.edu
Index

<<, 31
>>, 31
.*, 16
./, 16
[], 62
[2, 68, 127]
-, 21
', 16
~1, 68

absolute value, 25
adaptmesh, 42
abserror=, 45
cutoff=, 45
err=, 44
errg=, 44
hmax=, 44
hmin=, 44
inquire=, 45
isMetric=, 46
iso=, 45
keepbackvertices=, 45
maxsubdiv=, 45
metric=, 46
nbjacoby=, 45
nhsmooth=, 45
nbx=, 45
nomeshgeneration=, 46
omega=, 45
periodic=, 46
powerin=, 46
ratio=, 45
rescaling=, 45
splitin2, 46
splitpbedge=, 45
verbosity=, 45
alphanumeric, 21
append, 31
area, 23
area coordinate, 66
argument, 25
array, 22
FE function, 29
fespace, 56
initialisation, 188
mesh, 143
string, 192
sub, 189
average, 192
backward Euler method, 107
bamg, 36
BFGS, 79
bool, 22
border, 34
boundary condition, 68
break, 31
broadcast, 143
bubble, 60
CG, 63
CGNL, 187
checkmovemesh, 39
Cholesky, 63
Cint, 31
compiler, 21
concatenation, 44
connectivity, 70
continue, 31
dirrect, 119
cout, 31
Crout, 63
de Moivre’s formula, 25
discontinuous functions, 132
displacement vector, 93
divide
term to term, 16
domain decomposition, 124
dot product, 16
dumpTable, 24
dynamic load, 192

EigenValue, 104
eigenvector, 189
empty mesh, 39
endl, 31
exec, 24, 73

factorize=, 101
false, 22

FE function
[] : 62
    complex, 17, 28
    n, 62
    value, 62
FE space, 56
FE-function, 28, 56
FEspace
    (int, int ), 70
    ndof, 70
    nt, 70
fespace, 55
    P0, 55
    P1, 55
    P1b, 56
    P1dc, 55
    P1nc, 56
    P2, 56
    P2dc, 56
    periodic=, 63, 88
    RT0, 56

file
    am, 157, 158
    am_int, 36, 37, 157, 158
    amdba, 157, 158
    bamg, 36, 37, 155
    data base, 155
    ftq, 159
    mesh, 36, 37
    msh, 158
    nopo, 36, 37
    finite element space, 55
    fluid, 117
    for, 31
    formulas, 28
    func, 22

function, 100
tables, 38
functions, 26

GC, 68
GMRES, 63, 186
gnuplot, 73
grey=, 191

hat functions, 13
hTriangle, 23, 92, 187

ifstream, 31
include, 121
init=, 64
inside=, 69
int1d, 64
int2d, 64
intalledges, 64, 92, 187
interpolate, 68
    inside=, 69
    op=, 69
    t=, 69
isotropic, 94

jump, 92, 192

label, 23, 33, 34, 146
label=, 46
lagrangian, 41
lenEdge, 23, 92, 187
linearCG, 77
    eps=, 77
    nbiter=, 77
    precon=, 77
    veps=, 77
LinearCMRES
    eps=, 77
    nbiter=, 77
    precon=, 77
    veps=, 77
LU, 63

macro, 137, 189
matrix, 23, 67, 101
    =, 121
    complex, 68
    factorize=, 103
    full, 192
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpolate</td>
<td>68</td>
</tr>
<tr>
<td>load matrix</td>
<td>11</td>
</tr>
<tr>
<td>solver</td>
<td>68</td>
</tr>
<tr>
<td>solver=</td>
<td>121</td>
</tr>
<tr>
<td>stiffness matrix</td>
<td>11</td>
</tr>
<tr>
<td>matrix=complex</td>
<td>17</td>
</tr>
<tr>
<td>maximum</td>
<td>24</td>
</tr>
<tr>
<td>medit</td>
<td>73</td>
</tr>
<tr>
<td>mesh</td>
<td>22</td>
</tr>
<tr>
<td>3point bending</td>
<td>54</td>
</tr>
<tr>
<td>beam</td>
<td>50</td>
</tr>
<tr>
<td>Bezier curve</td>
<td>51</td>
</tr>
<tr>
<td>Cardioid</td>
<td>50</td>
</tr>
<tr>
<td>Cassini Egg</td>
<td>51</td>
</tr>
<tr>
<td>NACA0012</td>
<td>50</td>
</tr>
<tr>
<td>regular</td>
<td>42</td>
</tr>
<tr>
<td>Section of Engine</td>
<td>52</td>
</tr>
<tr>
<td>Smiling face</td>
<td>53</td>
</tr>
<tr>
<td>U-shape channel</td>
<td>52</td>
</tr>
<tr>
<td>V-shape cut</td>
<td>52</td>
</tr>
<tr>
<td>meshadapt</td>
<td></td>
</tr>
<tr>
<td>metric=</td>
<td>192</td>
</tr>
<tr>
<td>meshsplit</td>
<td>192</td>
</tr>
<tr>
<td>minimum</td>
<td>24</td>
</tr>
<tr>
<td>movemesh</td>
<td>39</td>
</tr>
<tr>
<td>mpirank</td>
<td>143</td>
</tr>
<tr>
<td>mpisize</td>
<td>143</td>
</tr>
<tr>
<td>Navier-Stokes</td>
<td>119</td>
</tr>
<tr>
<td>ndof</td>
<td>70</td>
</tr>
<tr>
<td>ndofK</td>
<td>70</td>
</tr>
<tr>
<td>Newton</td>
<td>79</td>
</tr>
<tr>
<td>NLCG</td>
<td>101</td>
</tr>
<tr>
<td>eps</td>
<td>77</td>
</tr>
<tr>
<td>nbiter</td>
<td>77</td>
</tr>
<tr>
<td>veps</td>
<td>77</td>
</tr>
<tr>
<td>normal</td>
<td>65</td>
</tr>
<tr>
<td>nt</td>
<td>70</td>
</tr>
<tr>
<td>nTonEdge</td>
<td>79</td>
</tr>
<tr>
<td>muEdge</td>
<td>23</td>
</tr>
<tr>
<td>muTriangle</td>
<td>23</td>
</tr>
<tr>
<td>ofstream</td>
<td>31</td>
</tr>
<tr>
<td>append</td>
<td>31</td>
</tr>
<tr>
<td>on</td>
<td>65</td>
</tr>
<tr>
<td>intersection</td>
<td>120</td>
</tr>
<tr>
<td>optimize=</td>
<td>68</td>
</tr>
<tr>
<td>P</td>
<td>23</td>
</tr>
<tr>
<td>P0</td>
<td>55</td>
</tr>
<tr>
<td>P1</td>
<td>55</td>
</tr>
<tr>
<td>P1-projection</td>
<td>13</td>
</tr>
<tr>
<td>P1b</td>
<td>56</td>
</tr>
<tr>
<td>P1dc</td>
<td>55</td>
</tr>
<tr>
<td>P1ne</td>
<td>56</td>
</tr>
<tr>
<td>P2</td>
<td>56</td>
</tr>
<tr>
<td>P2dc</td>
<td>56</td>
</tr>
<tr>
<td>periodic</td>
<td></td>
</tr>
<tr>
<td>plot</td>
<td></td>
</tr>
<tr>
<td>aspectratio</td>
<td>72</td>
</tr>
<tr>
<td>nbiso</td>
<td>72</td>
</tr>
<tr>
<td>border</td>
<td>34</td>
</tr>
<tr>
<td>cmm</td>
<td>188</td>
</tr>
<tr>
<td>bw</td>
<td>72</td>
</tr>
<tr>
<td>cmem</td>
<td>72</td>
</tr>
<tr>
<td>coef</td>
<td>72</td>
</tr>
<tr>
<td>cut</td>
<td>72</td>
</tr>
<tr>
<td>grey</td>
<td>72</td>
</tr>
<tr>
<td>mesh</td>
<td>34</td>
</tr>
<tr>
<td>nbarraw</td>
<td>72</td>
</tr>
<tr>
<td>ps</td>
<td>72</td>
</tr>
<tr>
<td>value</td>
<td>72</td>
</tr>
<tr>
<td>varrow</td>
<td>72</td>
</tr>
<tr>
<td>viso</td>
<td>72</td>
</tr>
<tr>
<td>precision</td>
<td>192</td>
</tr>
<tr>
<td>precon</td>
<td></td>
</tr>
<tr>
<td>problem</td>
<td>23</td>
</tr>
<tr>
<td>nKrylov</td>
<td>63</td>
</tr>
<tr>
<td>eps</td>
<td>63</td>
</tr>
<tr>
<td>init</td>
<td>64</td>
</tr>
<tr>
<td>precon</td>
<td>64</td>
</tr>
<tr>
<td>solver</td>
<td>63</td>
</tr>
<tr>
<td>tgv</td>
<td>64</td>
</tr>
<tr>
<td>186</td>
<td>191</td>
</tr>
<tr>
<td>processor</td>
<td>143</td>
</tr>
<tr>
<td>product</td>
<td></td>
</tr>
<tr>
<td>dot</td>
<td>16</td>
</tr>
<tr>
<td>term to term</td>
<td>16</td>
</tr>
<tr>
<td>qf2pT4P1</td>
<td>189</td>
</tr>
<tr>
<td>qforder</td>
<td>121</td>
</tr>
<tr>
<td>read files</td>
<td>37</td>
</tr>
<tr>
<td>readmesh</td>
<td>35</td>
</tr>
</tbody>
</table>

**Start of document**

```
interpolate, 68
load matrix, 11
solver, 68
solver=, 121
stiffness matrix, 11
matrix=complex, 17
maximum, 24
medit, 73
mesh, 22
3point bending, 54
beam, 50
Bezier curve, 51
Cardioid, 50
Cassini Egg, 51
NACA0012, 50
regular, 42
Section of Engine, 52
Smiling face, 53
U-shape channel, 52
V-shape cut, 52
meshadapt
metric=, 192
meshsplit, 192
minimum, 24
movemesh, 39
mpirank, 143
mpisize, 143
```

**End of document**
real, 22
region, 23, 131
Reusable matrices, 119
RT0, 56
RTortho, 188
savemesh, 35
schwarz, 143
sec:Plot, 71
shurr, 124, 126
singularity, 43
solve, 23, 63
eps=, 63
linear system, 16, 68
precon=, 64
solver=, 63
tgv=, 64, 68, 186, 191
solver=, 101
CG, 44, 63
Cholesky, 63
Crout, 63
GMRES, 63
LU, 63
UMFPACK, 63
split=, 46
square, 92, 187
Stokes, 117
stokes, 115
stop test, 63
absoulue, 122
strain tensor, 94
streamlines, 118
stress tensor, 94
string, 22
subdomains, 131
tanh, 189
Taylor-Hood, 120
transpose, 16, 30, 174
triangle
[ ], 36
label, 36
triangulate, 38, 190
true, 22
trunc, 46
label=, 46
tutorial
adapt.edp, 43
adaptindicatorP2.edp, 92
aTutorial.edp, 82
beam.edp, 95
blakschol.edp, 113
Fluidstruct.edp, 128
freeboundary.edp, 134
movemesh.edp, 41
NSUzawaCahouetChabart.edp, 121
periodic.edp, 88
periodic4.edp, 88
readmesh.edp, 37
Schwarz-gc.edp, 126
Schwarz-no-overlap.edp, 125
Schwarz-overlap.edp, 123
StokesUzawa.edp, 120
tutorial
convect.edp, 112
type of finite element, 55
UMFPACK, 63, 192
varf, 23, 64, 66, 121
variable, 21
variational formula, 13
veps=, 101
vertex
label, 36
x, 36
y, 36
while, 31
write files, 37
x, 23
y, 23
z, 23